UNIVERSITEIT ANTWERPEN
Faculteit Toegepaste Economische Wetenschappen
2004-2005

Tabu Search voor de Optimalisatie van Muzikale Fragmenten

Verhandeling voorgedragen
tot het bekomen van de graad
Handelsingenieur
in de Beleidsinformatica

Dorien Herremans

Promotor:
Dr. K. Sörensen

Voorwoord

Bij het maken van een wetenschappelijk werk is het steeds belangrijk om er plezier in te blijven hebben. Dit was zeker het geval bij deze thesis. Het onderwerp interesseert mij zeer sterk, omdat het twee van mijn grote passies combineert, namelijk wiskunde en muziek. Ik vond het enorm boeiend om ermee bezig te zijn.

Een thesis wordt natuurlijk nooit alleen geschreven. Ik wil daarom ook graag mijn promotor Kenneth Sörensen bedanken voor het nalezen en het beantwoorden van mijn vragen. Ook mijn vriendin Els is een enorme steun geweest en een hulp bij het nalezen. Tenslotte wil ik ook mijn ouders bedanken, die al jaren achter mij staan en mij de mogelijkheid hebben gegeven om handelsingenieur te studeren.

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Inleiding

Zal IBM de volgende Mozart ontwikkelen? Deze vraag lijkt ver van ons bed te staan. Er is echter heel wat aan de gang in de wereld van 'Computer Aided Composing', of kortweg CAC. Dit relatief nieuwe onderzoeksveld houdt zich bezig met het componeren van muziek door computers.

Wat maakt een muziekstuk beter dan een ander? Dit is een belangrijke vraag die we ons moeten stellen wanneer we het gebied van CAC verkennen. Er bestaat zoiets als 'muziektheorie', deze klassieke wetenschap legt verschillende regels op die bepalen wat een goed muziekstuk is. Wanneer we deze regels kunnen kwantificeren en er een belangrijkheid aan toekennen, dan krijgen we een soort van 'doelfunctie'. Een goed muziekstuk komt overeen met een hogere waarde van zijn doelfunctie. Dit doet ons sterk denken aan een optimalisatieprobleem. En inderdaad, er zijn reeds verschillende CAC-toepassingen ontwikkeld die gebruik maken van technieken uit de wereld van optimalisatie, zoals bijvoorbeeld genetische algoritmen.

In deze thesis gaan we enkele optimalisatietechnieken, namelijk metaheuristieken, meer in detail bekijken. We gaan ze vervolgens koppelen aan CAC. Hoe werkt een programma dat muziek kan componeren? Na enkele van deze voorbeelden te hebben besproken gaan we zelf een kleine toepassing ontwikkelen, gebaseerd op tabu search. We kozen voor tabu search omdat er in het verleden reeds veel projecten gebruik maakten van een genetisch algoritme. Van tabu search zijn er weinig of geen toepassingen bekend. We hopen hiermee aan te tonen dat een tabu search algoritme een nuttig instrument kan zijn bij het componeren van muziek.

In het eerste hoofdstuk gaan we bespreken hoe we het componeren van muziek kunnen beschouwen als een optimalisatieprobleem. We vertellen kort iets meer over muziektheorie en wat de 'optimalisatie van muziek' precies inhoudt. Vervolgens gaan we dieper in op de geschiedenis van computermuziek. Het lijkt voor de hand te liggen dat computers goed overweg kunnen met muziek, maar zoals zal blijken is dit niet altijd het geval geweest.

In hoofdstuk 2 vertellen we iets meer over de verschillende metaheuristieken. Dit zijn combinatorische optimalisatietechnieken die kunnen worden aangewend om complexe problemen op te lossen. De volgende metaheuristieken worden besproken: iterated local search, simulated annealing, genetische algoritmen en tabu search.

Hoofdstuk 3 koppelt de twee vorige hoofdstukken. We bespreken enkele concrete zaken zoals: hoe stellen we een muziekstuk voor in een computerprogramma? Welke programmeertalen zijn geschikt om CAC-toepassingen te schrijven? Daarenboven bespreken we enkele voorbeelden van CAC-programma's, namelijk GenJam en SIEDP.

Het tweede deel van deze thesis gaat over de toepassing die we zelf ontworpen hebben (TabuMusic). Het ontworpen programma werkt op basis van een tabu search en optimaliseert willekeurige homofone muziek. De werking wordt in detail uiteengezet in hoofdstuk 4. In hoofdstuk 5 maken we tenslotte een evaluatie van de resultaten.

De bijgevoegde cd-rom bevat de broncode van TabuMusic, samen met een gecompileerde versie voor win32 en linux. Het opensource programma iabc is ook bijgevoegd. Verder staan er ook enkele geoptimaliseerde muziekfragmenten op.

DEEL I

Theoretische Achtergrond

1 Muziek als optimaliserings-probleem

I dream of musical instruments obedient to my thought and which with their contribution of a whole new world of unsuspected sounds, will lend themselves to the exigencies of my inner rhythm.

Varese, 1937 zoals aangehaald in (Stephen, 2004)

1.1 Probleemstelling

Kunnen computers muziek componeren? Ja, dat kunnen ze zeker, we moeten de vraag misschien herformuleren als: "Kan een computer met behulp van optimalisatie-algoritmen een goed muziekstuk componeren?"

Eerst en vooral moeten we vaststellen dat een muziekstuk perfect voor te stellen is met behulp van een computer. Denk hiervoor maar aan digitale piano's, drumcomputers, sequencers en allerlei andere MIDI-toestellen[1]. Het gebruik van dit soort geluidssynthese is sterk ingeburgerd.

Hoewel muziek duidelijk is geassocieerd met emoties en artistieke expressie speelt logica er een niet te verwaarlozen rol in. Klassieke muziekstukken (en andere) zitten zeer logisch en wiskundig ineen en volgen over het algemeen de regels van de harmonie (Geeurickx, 1985). Een computer is in staat om met behulp van deze logica en regels zelf een muziekstuk te componeren.

[1]MIDI staat voor MIDI: Musical Instrument Digital Interface, een gestandaardiseerde interface gebruikt bij elektronische muziek-keyboards en PC's voor computer controle van muzikale instrumenten en apparaten (Disctronics, 2004). In sectie 1.4.3 wordt hierover meer gezegd.

Bestaat er een *optimaal* muziekstuk? Aan welke criteria moet muziek hiervoor voldoen? Wanneer we erin zouden slagen om objectieve criteria in verband met harmonie, melodie en ritme vast te leggen, dan kunnen we op basis hiervan een doelfunctie opstellen. Op die manier zijn we in staat om een 'score' toe te kennen aan muziek, die weergeeft hoe goed ze aan de criteria voldoet. Het is dan theoretisch mogelijk om een optimaal muziekstuk te 'berekenen'. Dit is dan een muziekstuk waarbij de doelfunctie zijn maximale waarde bereikt. In de praktijk blijkt het echter zeer moeilijk om deze criteria ook echt objectief vast te leggen (Kleeven en Philippi, 2002).

In de rest van dit hoofdstuk gaan we dieper in op het verband tussen muziek en optimalisatie. Verder bespreken we ook de geschiedenis van computermuziek en de begrippen die hierbij van belang zijn.

1.2 Muziek en optimalisatie

1.2.1 Definitie van muziek

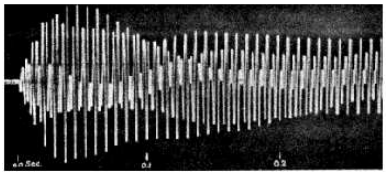
We komen in het dagelijkse leven bijna elke dag in contact met muziek. Als we over straat lopen, in de auto,... zelfs op het werk. Maar hoe wordt muziek gedefinieerd? Als we de term muziek opzoeken in de Encarta Encyclopedie, dan vinden we volgende definitie:

> Music is the organized movement of sounds through a continuum of time (Encarta, 1997).

Muziek is dus een opeenvolging van geluiden doorheen de tijd. Onze ervaring leert ons dat niet alle geluiden zomaar als muziek worden beschouwd. In de volgende sectie gaan we dieper in op de wiskundige aard van muziek. Waarom klinken bepaalde geluiden goed samen en andere niet?

1.2.2 Een wiskundige kijk op muziek

Muziek bestaat uit geluid. Geluiden zijn opgemaakt uit herhalende *geluidstrillingen*. De frequentie van deze trillingen wordt gemeten in Hertz (Hz), dit is het aantal trillingen per seconde (Land, 2003). In figuur 1.1 is de pianotoon C (129Hz) weergegeven. De luchtdruk is hier uitgezet tegen de tijd. In de figuur correspondeert de



Figuur 1.1: Grafische weergave van een pianotoon C van 129Hz (Biles, 1994)

toonhoogte met de frequentie, de luidheid met de amplitude en de klankkleur met de specifieke vorm van het golfpatroon.

In muziek zijn er slechts twee constante waarden. De eerste is de frequentie van de A (la) en bedraagt 440 Hz. De tweede constante waarde is de ratio van de frequenties tussen een halve toon, deze is $\sqrt[12]{2} = 1.0594630943593$. Met behulp van deze twee waarden kunnen de frequenties van alle mogelijke noten worden uitgerekend (Blackburn, 2004). Bijvoorbeeld:

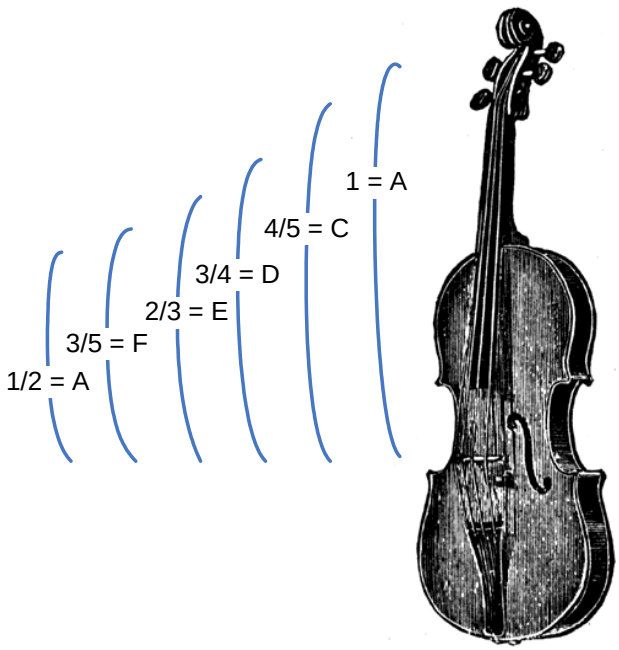
A$\sharp$ (1 halve toon hoger dan A)$= 440Hz \times \sqrt[12]{2} = 466.16$

Pythagoras was de eerste persoon die een verband zag tussen wiskunde en muziek. Hij was er sterk van overtuigd dat wiskunde overal was en dat elke waarde als een *verhouding* uit te drukken was. In de tijd van de Grieken was muziek nog niet zo ingewikkeld als vandaag en bestonden er maar vijf noten. Pythagoras wees erop dat elke noot kon gespeeld worden als een fractie van een snaar (Cox, 2004).

Stel dat een snaar klinkt als een A. Als we $\frac{4}{5}$ van de lengte van de snaar nemen, dan vinden we de volgende noot ($\approx C$). Verder vinden we: $\frac{3}{4} \approx D, \frac{2}{3} \approx E, \frac{3}{5} \approx F$ en $\frac{1}{2} \approx A$. Dit systeem wordt voorgesteld in figuur 1.2. Op deze manier kunnen we de vijf noten van de Grieken spelen. Men refereert hier soms naar als het 'Pythagorastoonsysteem' (van de Craats en Takens, 2001).

Hoe zijn we dan aan een systeem met twaalf noten geraakt? Men heeft de ratio's van Pythagoras op de andere noten toegepast. Zo vinden we bijvoorbeeld de noot B door $\frac{2}{3}$ de lengte van een E-snaar te nemen. Op deze manier vinden we twaalf noten. We kunnen dit grafisch weergeven met een spiraal, zie figuur 1.3. Langs de rand is er een schaalverdeling in graden. Deze schaalverdeling laat een logaritmische voorstelling toe.

Eén rondgang langs de linkse spiraal (360°) correspondeert met een stijging over zeven octaven. Eén octaaf meet dan $\frac{360}{7}$ graden. Een kwint, frequentieverhouding 2:3, meet dan $\frac{\log 3/2}{\log 2} \times \frac{360}{7} \approx 30.083786$ graden. De spaken zijn om de 30 graden

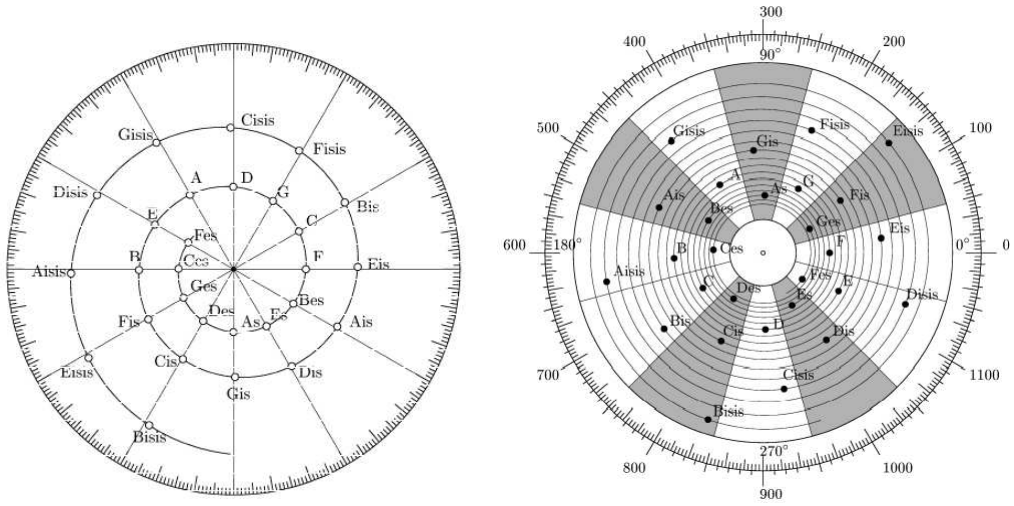


Figuur 1.2: De ratio's van Pythagoras

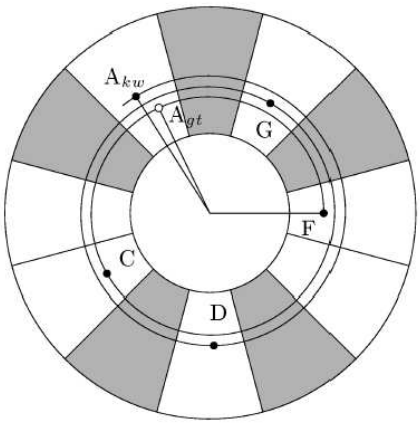
getekend. De spiraal aan de rechterkant in figuur 1.3 is strakker opgewonden. Eén rondgang rond de spiraal stelt nu één octaaf voor (in plaats van zeven). Daardoor is een kwint nu 210.5865 graden (van de Craats en Takens, 2001).

Er was echter een probleem met dit systeem. De frequenties van de noten die gevonden werden vertrekkend van een A waren niet helemaal identiek als deze gevonden vertrekkend van de A♯. Dit zorgde voor problemen wanneer twee verschillend gestemde instrumenten samen speelden (Cox, 2004). Vooral de grote tertsen (F–A, C–E en G–B) klinken vals. Neem bijvoorbeeld het interval F–A. In het systeem van Pythagoras kunnen we van de F naar de A gaan in vier intervallen (kwinten): F–C–G–D–A. Aangezien een kwint een interval is met verhouding $\frac{2}{3}$, verhouden de frequenties van F en A zich als $\frac{2^4}{3^4} = \frac{16}{81}$. Wanneer we de F twee octaven[2] hoger nemen wordt deze verhouding $\frac{16\times 2}{81} = \frac{64}{81}$. Wanneer we van dezelfde F vertrekken en we nemen de noot die een *zuivere grote terts* hoger ligt (dit is weer de A), dan moeten hun frequenties zich volgens Pythagoras verhouden als $\frac{4}{5} = \frac{64}{80}$. We merken op dat er een verschil zit tussen de eerste *valse* ($\frac{64}{81}$) en de tweede *zuivere* ($\frac{64}{80}$) grote terts (van de Craats en Takens, 2001). Dit kleine verschil in frequentie zorgt ervoor dat de geluidstrillingen gaan *zweven* en daardoor vals klinken (van de Craats en Takens, 2001). Figuur 1.4 geeft dit nog eens grafisch weer.

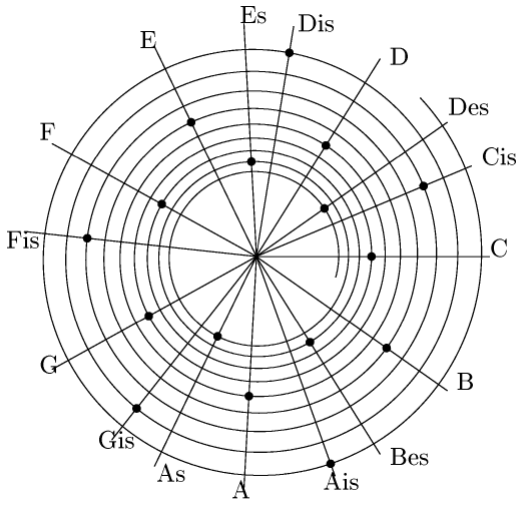
[2]Een octaaf hoger komt overeen met een verdubbeling in de frequentie



Figuur 1.3: De kwintenspiraal (van de Craats en Takens, 2001)



Figuur 1.4: De valse grote terts F–A_{kw} en de zuivere grote terts F–A_{gt} (van de Craats en Takens, 2001)



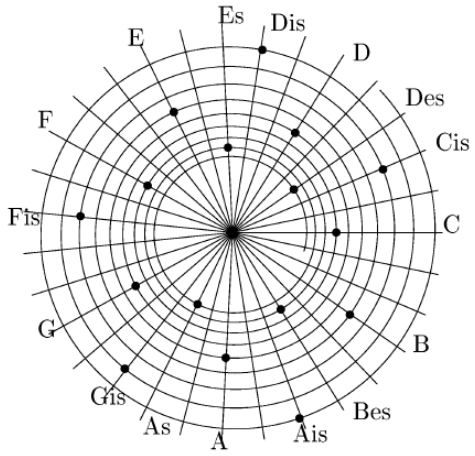
Figuur 1.5: De middentoonspiraal (van de Craats en Takens, 2001)

De *middentoonstemming* kwam met een oplossing voor dit probleem. Ze zorgde ervoor dat men gebruik maakte van de hierboven vernoemde constante ($\sqrt[12]{2}$) om de verhoudingen tussen de noten te bepalen. Op die manier klinken niet enkele noten heel vals, maar alle noten een heel klein beetje. We kunnen dit weer in spiraalvorm voorstellen, zie figuur 1.5. Zoals op de figuur blijkt is ook de middentoonstemming niet perfect. We zien dat Cis (C♯) en Des (D♭) lichtjes van elkaar verschillen. Als we naar een pianoklavier kijken, dan valt echter op dat er maar één toets is voor Cis en Des. Men zal dus moeten kiezen welk segment van 12 opeenvolgende noten men gebruikt uit de middentoonspiraal. In oudere muziek werd vaak gekozen voor het segment:

E♭–B♭–F–C–G–D–A–E–B–F♯–C♯–G♯

De kwint A♭–E♭ klinkt dan echter heel erg vals, hij is bijna een kwart toon te klein. De klank van de zwevingen die dit interval veroorzaken werd vroeger vergeleken met het huilen van een wolf, vandaar de naam *wolfskwint* (van de Craats en Takens, 2001).

Deze problemen worden opgelost door de *31-tonenstemming* van Huygens, zie figuur 1.6. Hij verbeterde de middentoonstemming door een veel fijnmaziger octaafverdeling toe te passen. Het resultaat is een wiskundig mooie oplossing, waarbij het octaaf in 31 delen is verdeeld. In figuur 1.6 zien we dit grafisch voorgesteld. De zuiverheid van de geproduceerde muziek zou veel beter zijn als dit systeem zou worden toegepast. In de praktijk is het echter niet zo gemakkelijk toe te passen, een normale



Figuur 1.6: De middentoonspiraal en het 31-tonensysteem van Huygens (van de Craats en Takens, 2001)

piano zou al snel 227 toetsen krijgen. In de hedendaagse muziek wordt daarom nog steeds de middentoonstemming gehanteerd (van de Craats en Takens, 2001).

Op basis van de frequenties van noten zullen sommigen beter bij elkaar passen dan andere (Bilotta, 2000). Een gevolg hiervan is dat er *toonladders*[3] ontstaan. Bepaalde combinaties van noten zullen 'droevige' muziek produceren, de zogenaamde 'mineur'-toonladders. In tegenstelling tot de 'diatonische' of 'majeur'-toonladders (Schell, 2002).

Een andere belangrijke consideratie zijn de *'harmonische tonen'*. Veel muzikanten vergeten dat wanneer we bijvoorbeeld een noot spelen op een viool, we niet alleen deze noot horen, maar ook een aantal harmonische tonen. Dit zijn noten met een proportionele frequentie aan de originele noot (Cox, 2004). Een voorbeeld maakt dit duidelijk:

Stel dat we een A spelen van 110 Hz. De harmonische tonen die hier samen mee klinken zijn voorgesteld in tabel 1.1. Deze tonen bepalen welke noten een goed akkoord vormen. Wanneer twee noten tegelijk worden gespeeld en hun harmonische tonen versterken elkaar, dan klinkt het akkoord aangenaam.

We hebben een korte beschrijving gegeven van de onderliggende aard van muziek. De regels van de harmonie zijn gebaseerd op hetgeen hierboven is beschreven, zonder dat zij daarbij gedetailleerd ingaat op frequenties en ratio's. Uit al deze mathematische theorie kunnen we concluderen dat we muziek kunnen beschouwen als geluidsgolven,

[3]Een toonladder is een op- of aflopende reeks van tonen in een vaste volgorde (Wikipedia, 2004).

	Noot	Frequentie (Hz)
Grondnoot	A	110
1ste harmonische toon	A (één octaaf hoger)	220
2de harmonische toon	E	330
3de harmonische toon	A (twee octaven hoger)	440
4de harmonische toon	C♯	550
5de harmonische toon	E	660

Tabel 1.1: Harmonische tonen van een A (Cox, 2004)

en meer bepaald als frequenties. Het lijkt nu duidelijk dat een hedendaagse computer geen moeite heeft met het manipuleren van zo'n simpele trillingen. De volgende sectie handelt over hoe we muziek kunnen beschouwen als een optimalisatieprobleem.

1.2.3 Optimalisatie van muziek

Wat is een optimalisatieprobleem? Als we het componeren van een muziekstuk willen bekijken als een optimalisatievraagstuk, dan is het belangrijk dat we eerst de eigenschappen hiervan bestuderen. Typisch herkennen we drie basiseigenschappen (Dréo et al., 2003):

1. Er zijn een aantal variabelen
2. We willen een doelfunctie maximaliseren (of minimaliseren)
3. Er zijn een aantal beperkingen

Wanneer we dit vertalen naar het optimaliseren van muziek, dan kunnen we deze eigenschappen als volgt invullen:

1. Er zijn een aantal variabelen. Welke noten bevinden zich waar in het muziekstuk? Welke duur hebben zij? Hoe lang is het muziekfragment?
2. Er bestaat een doelfunctie, we kunnen op een bepaalde manier een score toekennen aan een muziekstuk. Het is onze bedoeling om uiteindelijk het muziekstuk met de maximaal (of minimaal, naargelang de berekening) mogelijke score te vinden (Towsey et al., 2001).
3. Regels uit de muziektheorie zullen een aantal beperkingen opleggen. Zo zullen bepaalde combinaties van noten meer mogen voorkomen dan andere, of zal

er een bepaalde structuur in het ritme moeten zitten. Daarbij komt dat de gebruikte noten binnen een bepaald domein moeten liggen. Zo kan het domein bijvoorbeeld zijn: alle noten die het menselijk gehoor kan waarnemen of alle noten die door een bepaald instrument gespeeld kunnen worden (Towsey et al., 2001).

Zoals we reeds hebben opgemerkt is het concept 'optimale muziek' sterk afhankelijk van de manier waarop we een score toekennen. De gebruikte doelfunctie zal meestal bestaan uit volgende delen (Kleeven en Philippi, 2002):

- harmonische samenhang. Hoe goed klinken simultane noten samen?
- melodische samenhang. Hoe goed passen opeenvolgende noten bij elkaar?
- ritmische samenhang. Is het ritme consistent?

Er is echter geen universele manier ontwikkeld om dit te berekenen, en dat zal ook niet gebeuren in de toekomst. In muziek zit steeds een soort van persoonlijke inbreng. Deze inbreng kunnen we zeer moeilijk met de huidige muziektheorie beschrijven (Kleeven en Philippi, 2002). Bepaalde fragmenten zullen beter klinken wanneer ze niet aan de vooropgestelde regels voldoen. Dit kan te wijten zijn aan het feit dat de opgestelde regels nog niet volledig zijn. Het geven van een score aan muziek is op dit moment nog steeds zeer subjectief, en brengt veel discussies teweeg op het gebied van CAC.

De berekening van de doelfunctie zal ook sterk verschillen naargelang de muziekstijl (Brown, 2002). Dit komt doordat elke muziekstijl zijn eigen typische regels, ritmes en melodiën heeft. Wij zijn in staat verschillen tussen stijlen te herkennen, maar hoe doet een computer dat? Denk aan het volgende voorbeeld:

Stel dat een computer geprogrammeerd is met drie verschillende doelfuncties: één voor klassieke muziek, één voor popmuziek en één voor oosterse muziek. Wanneer we nu een klassiek muziekstuk ingeven en we vragen om de score te berekenen met behulp van de doelfunctie voor popmuziek zullen we hoogstwaarschijnlijk een slechte score krijgen. Terwijl het stuk misschien een meesterwerk uit de 19de eeuw is, bleek het ritme misschien te eentonig, te traag en de melodie te weinig samenhangend naar de normen van de popwereld. Met de doelfunctie voor oosterse muziek zouden we weer een slechte score krijgen. Wanneer we de doelfunctie voor klassieke muziek gebruiken, dan kunnen we pas een waarheidsgetrouwe score kunnen toekennen en het stuk evalueren.

In het verdere verloop van deze thesis zullen we het vooral hebben over de westerse klassieke muziek, tenzij het anders is aangegeven.

Wat betreft het 'optimaliseren' van muziek moeten we nog aangeven dat het niet onze bedoeling is om per sé de beste oplossing te vinden. Ondermeer omdat er in de vorige secties reeds is aangegeven dat het helemaal niet zeker is dat er zo'n oplossing bestaat. We willen aan de hand van bepaalde beperkingen een redelijk goed muziekfragment bekomen. In de volgende sectie vertellen we iets meer over een onderzoeksveld dat volop in bloei is, namelijk dat van 'Computer Assisted Composing' of CAC.

1.3 Computer Assisted Composing

Computer Assisted Composing (CAC) is een nieuw vakgebied dat de laatste jaren steeds belangrijker is geworden (Daubresse en Assayag, 2000). Het houdt zich bezig met het componeren van muziek met behulp van computers. We kunnen CAC opsplitsen in twee fundamentele delen (ThinkQuest, 1996).

1. Computers gebruiken om het componeren gemakkelijker te maken, als hulpmiddel.

2. Computers hun eigen muziek laten componeren.

In de *eerste betekenis* is de computer slechts een hulpmiddel (zij het dan een zeer krachtig) in het compositie-proces. Zo kan een componist bijvoorbeeld een computerprogramma gebruiken dat nagaat of zijn muziekstuk wel speelbaar is. Denk hierbij bijvoorbeeld aan een fragment voor gitaar. Het moet mogelijk zijn om de akkoorden die worden voorgesteld door de componist fysisch te spelen op het instrument. Daarvoor zal de computer alle mogelijke vingerposities moeten berekenen en controleren. Het zelfde probleem stelt zich ook bij andere instrumenten, zoals bijvoorbeeld de piano (Viana en de Morais Júnior, 2003). In sectie 3.4 wordt hiervan een voorbeeld behandeld, namelijk SIEDP.

In de *tweede betekenis* wordt de computer zelf componist. Het is vooral deze betekenis die in de literatuur op dit ogenblik het meeste aandacht krijgt. Het principe van deze benadering kunnen we meestal in het volgende raamwerk passen (Jacob, 1995).

1. Geef criteria op die bepalen wat een goed muziekstuk is.

2. Bereken een zo optimaal mogelijk muziekstuk.

Een niet te onderschatten moeilijkheid ligt bij de keuze van de berekeningswijze. Hoe vinden we een goed fragment wanneer er zoveel mogelijke combinaties zijn. We kunnen niet zomaar alle mogelijke muziekstukken proberen en zien welke het beste is. In dat geval zouden we enorm lang bezig zijn. Er bestaan betere en meer gestructureerde methodes om gericht naar een beter muziekfragment te zoeken, bijvoorbeeld genetische algoritmen (Jacob, 1995) en cellular automata (Miranda, 2001). Hierop komen we later nog uitgebreid terug wanneer we het project GenJam bespreken in hoofdstuk 3.

We zullen ons vooral focussen op de tweede betekenis van CAC: het gebruik van de computer om muziek te componeren. De nadruk ligt in deze thesis vooral op de algoritmen die gebruikt worden om een goed muziekstuk te vinden. Het is niet onze bedoeling om een cursus muziektheorie samen te vatten, toch is het interessant om kort iets te zeggen over de structuur van muziek en een schets te geven van de geschiedenis van computermuziek.

1.4 De geschiedenis van computermuziek

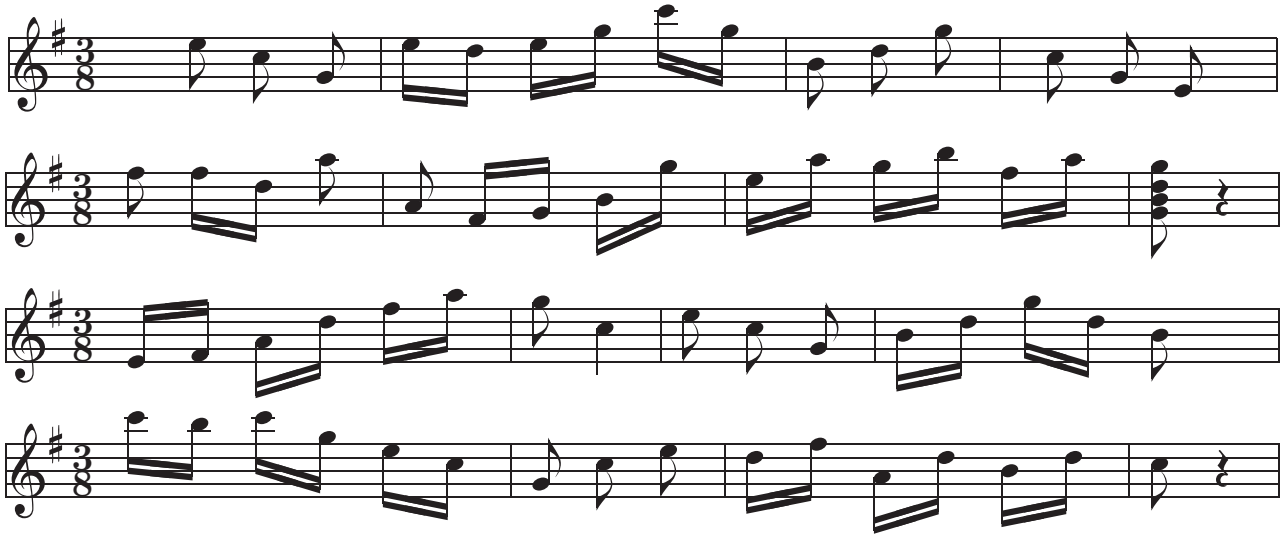
1.4.1 Vroege geschiedenis van klassieke muziek

Om een algemeen beeld te krijgen van het ontstaan van klassieke muziek, schetsen we hier de evolutie van klassieke muziek zonder al te veel in details op te gaan. We beginnen in de oudheid en eindigen in het heden.

In de **oudheid** begint het verhaal van klassieke muziek. De belangrijkste invloed van de oudheid ligt op het gebied van de *muziektheorie*. Ondermeer de ratio's van Pythagoras zitten hier voor iets tussen. Dit is eerder besproken in sectie 1.2.2 (Wikipedia, 2004).

Vanaf de **vroege Middeleeuwen** (tot 1000) is het vooral de kerk die een grote invloed uitoefent op de klassieke muziek. De melodieën gezongen in de kerk waren *eenstemmig* en hadden vaak een *Aziatische oorsprong* (Zychowicz, 2005).

In de **Middeleeuwen** (1000-1450) is *meerstemmigheid*, ook wel polyfonie genoemd, een belangrijke vernieuwing. Twee of meer instrumenten spelen hierbij tegelijk.



Figuur 1.7: Muziekstuk gegenereerd met het dobbelsteenspel van Mozart (Boskamp, 1997)

Geleidelijk aan werd een systeem van *muzieknotatie* ontwikkeld waarbij de noot wordt voorgesteld als een punt op een balk met lijnen (Zychowicz, 2005).

Tijdens de **Renaissance** (1450-1600) werd naast religieuze muziek steeds meer *profane en instrumentale muziek* gemaakt. Er komen strengere regels omtrent de correcte samenklank, ofwel *consonantie* van noten. Tenslotte gaat men meer aandacht besteden aan de relatie tekst-muziek (Wikipedia, 2004).

De **Barok** (1600-1750) zorgde voor een verandering van muziekstijl op vijf jaar tijd[4]. De meeste *moderne muziekinstrumenten* werden ontwikkeld in de Barok: de strijkinstrumenten en blaasinstrumenten. De Barok eindigt met de dood van de componist Johann Sebastian Bach (1685-1750) (SonyMusicEntertainement, 2001).

Het **Classicisme** (1750-1810) geeft haar naam aan de klassieke muziek. Deze korte periode wordt gekenmerkt door componisten zoals Wolfgang Amadeus Mozart en Joseph Haydn. Nieuwe vormen van muziek zoals de *sonatevorm* en de *symfonie* ontstaan in het Classicisme (Wikipedia, 2004).

De componist Mozart heeft in deze periode reeds een bijdrage geleverd aan de wereld van het CAC. Mozart is namelijk de uitvinder van het 'musikalisches würfelspiel', ofwel *'muzikaal dobbelsteenspel'*. Dit spel laat toe om willekeurige muziekstukken te genereren met behulp van twee dobbelstenen en 12 korte muziekfragmenten (met

[4]Dit komt ondermeer omdat de *monodie*, met het systeem van *'basso continuo'* opkomen, samen met de harmonie.

als lengte 1 maat) (Burns, 2005). Zijn idee was dat de spelers in staat waren een groot aantal melodiën te genereren door willekeurig te selecteren uit een aantal keuzemogelijkheden (Peterson, 2001). Voor het componeren van een muziekstuk van 16 maten gaat het spel als volgt (MathsoftEngineering en Education, 2004):

Doe voor elke maat m van 1 tot 15 (exclusief $m = 8$) het volgende:

1. Gooi de dobbelstenen zodat je een getal i krijgt dat elf mogelijke waarden kan aannemen (van 2 tot en met 12).

2. Kies als opvulling van maat m het fragment dat bij i hoort

Deze eenvoudige regel geldt voor elke maat, behalve voor maat 8 en 16. Hiervoor zijn enkel fragment 2 of 3 mogelijk. De beslissing van welk fragment we nemen zal afhangen van het feit of het gegooide getal i even of oneven is. De reden voor deze laatste regel ligt in de muziek die we genereren. Typisch klinkt deze vrij sonateachtig en zorgen de 8ste en 16de maat voor een rustpunt (zie figuur 1.7). Deze methode op zich is redelijk simpel en produceert toch goede muziek. Een niet te verwaarlozen voorwaarde is echter het hebben van de 12 korte fragmenten waarop alles is gebaseerd. Mozart heeft er een heleboel achtergelaten en wanneer we het spel hierop toepassen krijgen we een verrassend muziekje met een duidelijke 'Mozart-tint'. Een mogelijk resultaat van het dobbelsteenspel is weergegeven in figuur 1.7.

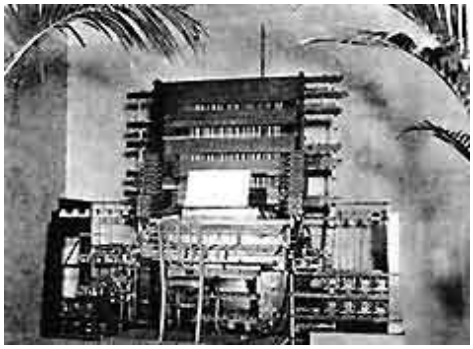
Een volgende periode is de **Romantiek** (1810-1910). Deze muziek maakt ongeveer 95% uit van wat op dit moment in concerten te horen is. De Verlichtingsgedachte stimuleerde diverse muzikale ontwikkelingen. Dit leidde tot complexe *harmonische ontwikkelingen*, steeds verbeterende *muziekinstrumenten*, grotere muziekstukken, enzoverder (Zychowicz, 2005).

Vanaf de **20ste eeuw** bestaan er verschillende stromingen gelijktijdig: het Modernisme, Neo-stijlen zoals het Neo-Calassicisme en Neo-Barok, Avant-Gardise, Serialisme, Minimal music, New Complexity, Nieuwe Eenvoud en Post Modernisme. We beperken ons tot een opsomming van deze stijlen, een volledige bespreking zou ons te ver leiden (Blood, 2004).

1.4.2 Nieuwe instrumenten en machines

Muziekinstrumenten bestaan al heel lang. Ze worden in de loop van de geschiedenis voortdurend bijgewerkt en verfijnd. In het begin van de 20ste eeuw beginnen

mensen echter een ander soort instrumenten te ontwikkelen, de *elektronische muziekinstrumenten*. In 1906 registreert Thaddeus Cahil in de Verenigde Staten een patent op een 'elektrisch gebaseerd geluidsgeneratie systeem', beter gekend als de *'dynamofoon'* of het 'telharmonicum' (tadream.net, 2005), zie figuur 1.8. Hoewel de

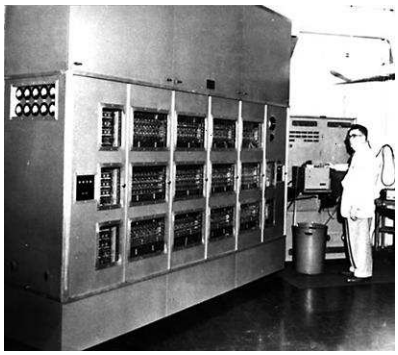


Figuur 1.8: De dynamofoon (Sonhors, 2005)

afmetingen van het apparaat indrukwekkend waren (200 ton gewicht en 1.8 meter hoog), kreeg de uitvinding niet veel aandacht. Deze machine was het eerste additieve geluidssyntheseapparaat en kon geluid produceren van verschillende frequenties en intensiteit. In het begin van de 20ste eeuw werden meerdere van deze syntheseapparaten gebouwd door onder meer Theremin, Milhaud, Mager en Martenot. Deze 'instrumenten' waren allemaal in staat om geluiden te creëren die vroeger ondenkbaar waren. Helaas waren ze geen groot succes. Een eerste reden is dat ze allemaal een beperkte capaciteit hadden en vrij primitief gebouwd waren. Verder waren er nog geen goede methodes om de geproduceerde geluiden op te slaan en te bewerken. Ten slotte wordt vaak beweerd dat de muziekwereld nog niet klaar was voor instrumenten die een radicale verandering teweeg brachten. Vele componisten bleven de klassieke Westerse instrumenten trouw (Stephen, 2004).

In 1947 was er een doorbraak op het gebied van CAC. Lejaren Hiller en Leonard Isaacson experimenteerden met de Illiac I'-mainframe, een Univac computer (figuur 1.9), aan de universiteit van Illinois (Burns, 2005). Met hun onderzoek wilden ze aantonen dat (Pearce et al., 2002):

- muzikale technieken kunnen worden verwerkt door computerprogrammering.
- computers kunnen worden gebruikt door componisten om hun compositietechnieken uit te breiden.
- computers op een ongebruikelijke manier kunnen worden ingezet om een radi-



Figuur 1.9: De Illiac-mainframe (MediaArtNet, 2004)

caal ander soort muziek te bekomen.

Hun experiment bestond uit vier delen, met toenemende complexiteit (Kleeven en Philippi, 2002):

- experiment 1: generatie van muziek met een beperkt aantal regels.
- experiment 2: generatie van muziek met waarbij telkens meer regels werden toegevoegd.
- experiment 3: implementatie van ritme, dynamiek en tempo.
- experiment 4: generatie van muziek op basis van Markov ketens. Aan de hand van alle voorgaande noten werd de kans berekend dat de volgende noot bijvoorbeeld een C zou zijn. De noot met de hoogste kans werd gekozen.

Ze waren de eersten die erin slaagden om een computer een muziekstuk volledig zelf te laten componeren. Het resultaat is de bekende 'Illiac Suite', ook bekend als 'String Quartet No. 4', uitgebracht in 1956 (ThinkQuest, 2004). We moeten opmerken dat hoewel de muziek werd geschreven door de computer, het stuk werd gespeeld door echte instrumenten. Met andere woorden, de Illiac I was niet in staat om de muziek te genereren (MediaArtNet, 2004).

Op het vlak van *elektronische muziek* beginnen er echter serieuze veranderingen te komen vanaf 1948. Pierre Schaeffer was de eerste die er begon te experimenteren met elektronische muziek. Hij nam geluiden op uit het alledaagse leven, bewerkte en mixte ze. Zijn 'Musique Concrete' was het eerste muziekstuk dat werd opgevoerd zonder dat er mensen aan te pas kwamen. Schaeffer kreeg vele volgelingen. Op deze manier begon het elektronische muziektijdperk echt (Stephen, 2004).

1.4.3 Computers en muziek

Een van de eerste echte *computerprogramma's* voor muziek werd geschreven in de jaren '60 door Max Mathews (Stephen, 2004). Matthews werkte bij Bell Labs, een onderzoekscentrum dat door AT&T werd gerund. Hij maakte er voorwerpen voor het telefoonbedrijf, zijn vrije tijd gebruikte hij om muzieksoftware te ontwerpen. De software die hij schreef heette 'MUSIC'. In het begin was het programma in staat om enkele tonen te produceren. De volgende versie, MUSIC II, kon vier tonen tegelijk produceren. Op een van de snelste computers van die tijd koste het echter nog ongeveer een uur computertijd om twee minuten muziek te genereren. De volgende versies, MUSIC IV en V waren efficiënter en konden nog meer geluiden produceren (tadream.net, 2005). Steeds meer componisten begonnen zich te vertonen in het labo van Matthews. Uiteindelijk werden er zelf enkele componisten als assistent aangenomen en werd het project een volwaardig onderzoeksproject (UCSC, 2004).

In de jaren '70 werden er een heleboel onderzoeksgroepen opgericht die computermuziek bestudeerden. De bekendste waren die van M.I.T., De Universiteit van Illinois in Champaigne-Urbana, de Universiteit van California in San Diego en het 'Institut de Recherche et Coordination Acoustic/Music' in Parijs (IRCAM) (UCSC, 2004). Deze centra werkten allemaal met een enorme mainframe. Een muziekstuk componeren was toen een langdurig proces. Na enkele dagen tekst ingeven, compileerde de computer de geluiden op een cassette of harde schijf. Vervolgens moest dit nog eens door een digitaal-analoogconverteerder worden overgezet op een cassette. Dan pas kon men de muziek horen. In deze tijd werden er dan ook meer papers geschreven dan muziek gecomponeerd (UCSC, 2004).

In 1968 ontwikkelt Max Mathews een van de eerste *hybride systemen*. Dit wil zeggen dat hij een koppeling maakte tussen een computer en een synthesizer. Terwijl een muzikant op een synthesizer speelde nam het GROOVE programma van Mathews alles op. Daarna kon deze muziek bewerkt worden door de computer en terug afgespeeld worden door de synthesizer. Andere onderzoekers hebben enkele variaties hierop gemaakt (CSounds, 2004).

Het grote succes van hybride systemen komt samen met de komst van de eerste thuiscomputers. De Apple II maakte het bijvoorbeeld bijzonder gemakkelijk om een synthesizer aan te sluiten. Een breed publiek kwam nu in contact met de wereld van computercompositie. Ook de komst van de Commodore 64, in 1982 was een doorbraak. Dit type computer bevatte een vierstemmige synthesizer op één chip

(UCSC, 2004).

Tegen 1983 kwam de *MIDI-standaard* (Musical Interface Digital Instrument) er. Deze standaard maakt het mogelijk om gelijk welke computer met eender welke synthesizer te verbinden. De verkoop van MIDI-synthesizers kende een enorme 'boom' in de jaren '80 (Glatt, 2004). Het MIDI-systeem is een communicatieprotocol voor elektronische muziek dat zowel hardware als software combineert. Een belangrijk concept hierbij is een *MIDI-event*. Een event stelt de ontvangst (of verzending) van een aantal bytes voor. Twee veel voorkomende events zijn (Wiggins et al., 1993a):

- `Note On`
- `Note Off`

Elke event bestaat steeds uit drie delen (Wiggins et al., 1993a):

- een identificeerder, of ID
- een toonhoogte (uitgedrukt in een integer-waarde)
- het tijdstip waarop de noot wordt gespeeld

We kunnen al deze informatiestromen bundelen in een bestand, de zogenaamde MIDI-file. Hierin zit alle informatie vervat om een fragment te spelen op een synthesizer of MIDI-instrument. Aangezien we een MIDI-bestand niet in ASCII-vorm (tekstvorm) kunnen lezen, hebben we gebruik gemaakt van het conversieprogramma 'MIDI File DisAssembler'. Een tool die gratis kan afgeladen worden op het net (Glatt, 1998). De belangrijkste functie van het programma is dat het een MIDI-bestand kan omzetten in een leesbaar ASCII-bestand. De tekstversie van een willekeurig gekozen muziekstuk is gegeven in bijlage A.

In de tekstuele vorm van het MIDI-bestand zien we de drie eigenschappen van een event duidelijk terugkomen:

- ID: de nummer voor een event
- toonhoogte of 'pitch': E 4 voor het eerste event → een E (mi) in het vierde octaaf
- de snelheid van indrukken: die blijft hier onveranderlijk en bedraagt nul.

In het begin van het bestand wordt additionele informatie gegeven over het muziekstuk. Bijvoorbeeld de toonaard, het tempo, de maatsoort, ...

In de onderzoekscentra van computermuziek focuste men zich op het ontwikkelen van software. De meeste MIDI-instrumenten waren eigenlijk gewoon heel simpele computers die een bepaald programma draaiden. Op M.I.T. werd een versie van het MUSIC programma gemaakt die kon worden gecompileerd op bijna alle platformen. De mainframes werden vervangen door desktops en niet lang daarna werd de taal CSound ontwikkeld. Dit is een zeer krachtige taal die nu vaak wordt gebruikt voor het ontwikkelen van compositie-programma's (Stephen, 2004). Ondertussen zijn er al meer talen en toepassingen ontwikkeld (zie hoofdstuk 3. De wereld van CAC krijgt steeds meer mogelijkheden ...

In het volgende hoofdstuk gaan we dieper in op een bepaalde soort optimalisatie-technieken, namelijk metaheuristieken.

2 Metaheuristieken

If you can't solve a problem, then there is an easier problem you can solve: find it.

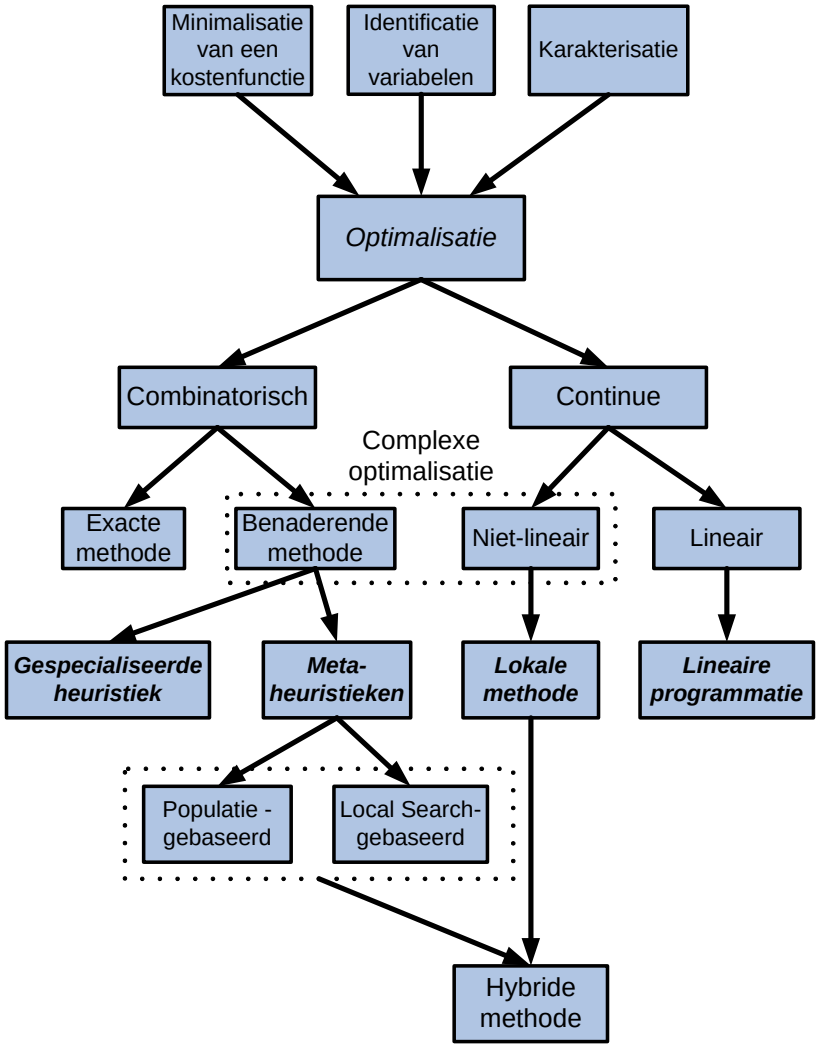
(Polya, 1957)

2.1 Context

Metaheuristiek is een vakgebied in opbloei. Dit is onder meer te wijten aan de steeds grotere rekenkracht van computers. Bepaalde metaheuristieken die vroeger niet te implementeren waren, kunnen nu gemakkelijk worden verwerkt door computers. Een precieze definitie en omschrijving van deze methodes wordt gegeven in sectie 2.3. Metaheuristieken zijn optimalisatietechnieken die vaak gebaseerd zijn op processen uit andere vakgebieden die niets te maken hebben met optimalisatie, zoals biologie en fysica. Dit zal blijken wanneer we de verschillende metaheuristieken verder gaan bespreken (Pirlot, 1996).

Voor welke problemen zijn metaheuristieken de geschikte oplossingsmethode? Een voorstelling van de verschillende optimalisatieproblemen en de manier waarop ze worden opgelost wordt gegeven in figuur 2.1.

In figuur 2.1 vertrekt men van de basiseigenschappen van optimalisatieproblemen: ze minimaliseren steeds een doelfunctie, hebben een aantal (doel-)variabelen die geïdentificeerd moeten worden en hebben een aantal beperkingen. Deze drie eigenschappen karakteriseren het volledige probleem. We kunnen vervolgens een grote opdeling maken tussen discrete en continue problemen. *Discrete problemen* zijn problemen waarbij de uitkomstenverzameling 'telbaar' is, we kunnen de uitkomsten als het ware tellen (Willemé en van Rompay, 2001-2002). Deze verzameling kan een



Figuur 2.1: Algemene classificatie van optimalisatiemethodes (Dréo et al., 2003)

eindig of oneindig aantal elementen bevatten. In het *continue* geval is de uitkomstenverzameling onaftelbaar en bevat zij steeds een oneindig aantal elementen (van Nuffelen, 1998).

Bij de discrete problemen interesseren we ons vooral voor de *combinatorische* problemen. Dit zijn optimaliseringsproblemen die de beste combinatie van een aantal variabelen of variabele-waarden proberen te vinden (Wikipedia, 2004). Een voorbeeld van een discreet probleem is het bekende *Travelling Salesman Problem* (TSP). Een handelsreiziger moet n steden exact één keer bezoeken. In zijn eenvoudigste vorm bestaat het TSP probleem uit het minimaliseren van de totale reisduur, waarbij de afstanden tussen de steden gekend zijn. Dit probleem kan nog worden uitgebreid door bijvoorbeeld een beperking op de tijd waarop hij een bepaalde klant moet bezoeken, het aantal 'folders' dat hij kan meenemen, enz. We komen later nog uit-

gebreid op dit voorbeeld terug. In bepaalde gevallen kan dit soort probleem een berekening vergen waarvan de grootte exponentieel toeneemt met het aantal steden. De volgende illustratie maakt ons het belang van exponentiële groei duidelijk:

> Een oude legende vertelt het verhaal van de uitvinder van het schaakspel en zijn opdrachtgever, de koning van Persië. Naar verluidt was de koning zeer tevreden met dit nieuwe spel en daarom wou hij de uitvinder een weelderige compensatie geven. De man zou genoeg eten krijgen om zijn familie een jaar te voeden. De uitvinder, een wiskundige begaafd man, bedacht een slimme manier om meer te krijgen.
>
> Hij kwam tot een akkoord met de koning. Op de eerste dag zou de koning één rijstkorrel op het eerste vakje van het schaakbord leggen voor de uitvinder. De tweede dag zou hij dubbel zoveel rijstkorrels op het tweede vakje leggen. De derde dag dubbel zoveel op het derde vakje, enzovoorts tot het 64ste vakje (Bick, 2000).
>
> De koning vond dit een fantastisch idee en hij liet zijn bedienden de eerste dag een korrel rijst klaarleggen voor de uitvinder. Zoals afgesproken kwam de man elke dag de klaargelegde rijstkorrels halen om zijn gezin te voeden. De eerste dagen noemde iedereen hem een idioot en vreesde zijn gezin zelfs voor ondervoeding.
>
> Maar het tij keerde. Het aantal rijstkorrels die hij mocht komen afhalen verdubbelde elke dag en tegen de negende dag waren het er al 256 (2^9), op de tiende 512, op de elfde 1024, 2048 op de twaalfde dag ... Tegen de 64ste dag was de koning meer dan 10 000 000 000 000 000 000 of 10^{19} rijstkorrels verschuldigd aan de uitvinder. Om zo'n hoeveelheid rijst de groeien is een oppervlakte nodig die groter is dan de aarde, inclusief de oceanen.
>
> De koning kon deze vraag niet voldoen en eindigde als schuldenaar, terwijl de uitvinder van het schaakspel en zijn familie hun intrek namen in het paleis (Dittli, 2003).

Zo'n exponentiële groei van de rekentijd willen we zeker vermijden. Om dit te bereiken zijn er benaderende methodes zoals specifieke heuristieken en metaheuristieken. Een belangrijke eigenschap van deze technieken is dat ze op een relatief korte tijd tot een redelijk goede oplossing kunnen komen, in plaats van jaren te rekenen om de allerbeste oplossing te vinden.

In het *continue* geval maakt men een onderscheid tussen lineaire en niet-lineaire problemen. Lineaire problemen kunnen worden opgelost met behulp van lineair pro-

grammeren. Niet lineaire problemen zijn moeilijk op te lossen. We kunnen gebruik maken van lokale methodes, of ze discretiseren en met behulp van metaheuristieken oplossen (Dréo et al., 2003).

Samenvattend kunnen we stellen dat metaheuristieken zullen worden gebruikt in twee gevallen:

- discreet: wanneer de omvang van het probleem exponentieel toeneemt. We kunnen theoretisch gezien wel tot de beste oplossing komen, maar dit zal enorm veel rekentijd in beslag nemen.
- continu: wanneer er geen algoritme gekend is dat naar een globaal optimum leidt na een beperkt aantal bewerkingen.

In de literatuur rond optimalisatie en artificiële intelligentie bestaan er een aantal klassieke problemen. Zo is er bijvoorbeeld het al eerder besproken travelling salesman problem, het knapsack-probleem, het vehicle routing probleem,... We gaan deze problemen kort schetsen, zodat we in de volgende hoofdstukken de draad terug kunnen oppikken en concrete oplossingsmethodes voorstellen voor een aantal van deze complexe problemen. We merken op dat ze slechts metaforen zijn. We kunnen andere problemen met behulp van deze klassieke problemen interpreteren en zo gebruik maken van de oplossingsmethoden die worden voorgesteld.

Travelling salesman problem

De term 'Travelling Salesman' is voor het eerst gebruikt in 1932 in het Duitse boek: "The travelling salesman, how and what he should do to get commissions and be successful in his business". Dit boek is geschreven door een veteraan-handelsreiziger (Michalewicz, 1992). In 1948 heeft RAND Corporation dit probleem opnieuw geïntroduceerd. Nadien is het een zeer bekend en populair probleem geworden, onder meer door de opkomst van technieken zoals lineair programmeren (Michalewicz, 1992).

Het travelling salesman problem (TSP) handelt, zoals reeds blijkt uit de naam, over een handelsreiziger. Deze moet n steden exact één maal bezoeken en dan terugkeren naar zijn vertrekpunt. Hoe moet hij zijn reis plannen, gegeven de kost om tussen alle steden te reizen (Luger en Stubblefield, 2003)? Dit interesseert ons omdat het aantal mogelijke oplossingen exponentieel toeneemt. Het zal dus zeer lang duren om de allerbeste volgorde van steden te vinden (Corne et al., 1999).

Knapsack problem

Het 'knapsack problem' of rugzakprobleem is een ander voorbeeld van een klassiek optimalisatieprobleem. We hebben een verzameling van n items, die allemaal een bepaald gewicht en een winst hebben. Het probleem stelt zich wanneer we een rugzak met een capaciteit C optimaal willen vullen. Onze doelstelling is het vinden van een deelverzameling die optimaal is, waarbij de winst maximaal is. Dit terwijl het totale gewicht de capaciteit C niet overschrijdt, zodat de oplossing geldig blijft. Een geldige oplossing wordt ook 'feasible' genoemd (Sörensen). Een voorbeeld van gegevens voor een knapsack probleem wordt gegeven in tabel 2.1 (zie verder). Dit wordt besproken in sectie 2.3.3.

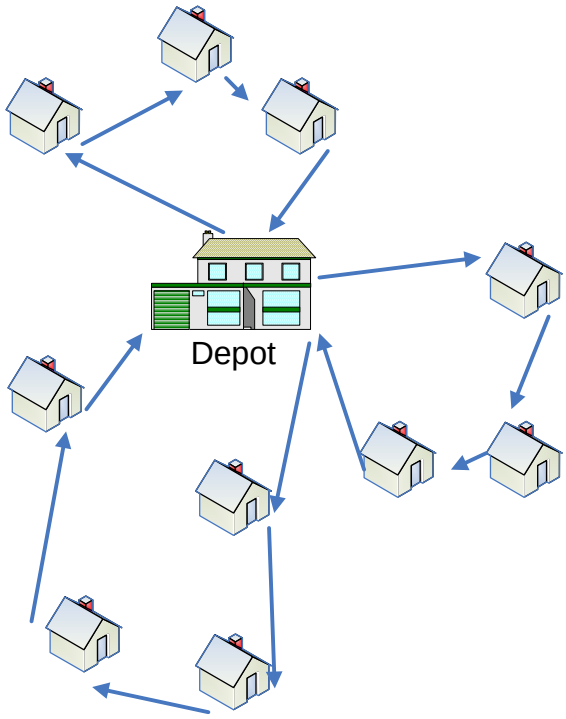
Vehicle routing problem

Bij het 'vehicle routing problem', of simpelweg VRP, hebben we een vloot van voertuigen gestationeerd in een of meerdere depots. Er zijn ook een aantal geografisch verspreide steden, met elk een gekende vraag, die moeten worden bezocht. De opgave is om de verzameling van routes, elk vertrekkend en eindigend in een depot, te optimaliseren. Dit wil zeggen dat de kost minimaal moet zijn. In figuur 2.2 wordt een voorbeeld gegeven van een mogelijk VRP probleem met één depot (Mastrolilli, 2005).

Dit soort problemen komen veel voor in de praktijk. Bijvoorbeeld op het gebied van logistiek, bij transportfirma's en in de distributiesector. Zo heeft de firma 'IJsboerke' enkele jaren geleden een systeem geïmplementeerd dat elke morgen de routes van de koelwagens berekende. We herkennen hier duidelijk een VRP systeem (hetgeen zij 'Home Vending Support Application noemen'): er zijn een aantal depots waar koelwagens kunnen vertrekken, er zijn een aantal klanten met een gekende vraag (of toch een voorspelbare vraag) en vertrekkend van deze gegevens wil men de kortste route bepalen (Selligent, 2005).

In de praktijk kunnen er natuurlijk nog extra beperkingen bijkomen. Enkele veelvoorkomende worden hier opgesomd (VRPWeb, 2005):

- een beperking op de capaciteit van een voertuig. Bijvoorbeeld: een vrachtwagen kan maar maximum 10 ton vervoeren. Dit probleem is gekend als 'Capacitated VRP' of CVRP (Gendreau et al., 2001).



Figuur 2.2: CVRP met één depot, op basis van (Corne et al., 1999)

- elke klant moet bediend worden binnen een bepaald tijdsvenster, ook wel bekend als 'VRP with time windows' of VRPTW (Corne et al., 1999).
- er zijn meerdere depots, 'Multiple Depot VRP' of MDVRP.
- klanten kunnen goederen terug meegeven naar het depot, 'VRP with Pick-Up and Delivering of VRPPD.
- klanten kunnen bediend worden door meerdere voertuigen, 'Split Delivery VRP' of SDVRP.
- ...

2.2 Heuristiek

Heuristiek is de kunst van het ontdekken en uitvinden. Het woord heuristiek stamt af van het Griekse woord 'heuriskein' ($\varepsilon\upsilon\rho\iota\sigma\kappa\varepsilon\iota\nu$), wat 'vinden' betekent (Wikipedia, 2004). Denk maar aan de kreet die Archimedes riep toen hij uit zijn befaamde badkamer kwam, 'Eureka', wat zoveel betekent als 'Ik heb het gevonden'.

Figuur 2.3: George Polya (CIS, 2004)

Een van de personen die meegeholpen heeft om heuristiek populair te maken in de 20ste eeuw is George Polya. In 1957 schreef hij een boek (Polya, 1957), waarin hij de verschillende probleemoplossende methodes beschrijft. Hij omschrijft heuristiek daarin als: "the study of the methods and rules of discovery and invention". Als kind was hij gefascineerd door de manier waarop wiskundigen bewijzen ontwikkelden. Hij leerde op school de bewijzen, maar hij leerde niet hoe deze tot stand zijn gekomen. In zijn boek stelt hij volgende stappen voor om een wiskundig probleem op te lossen (Wikipedia, 2004):

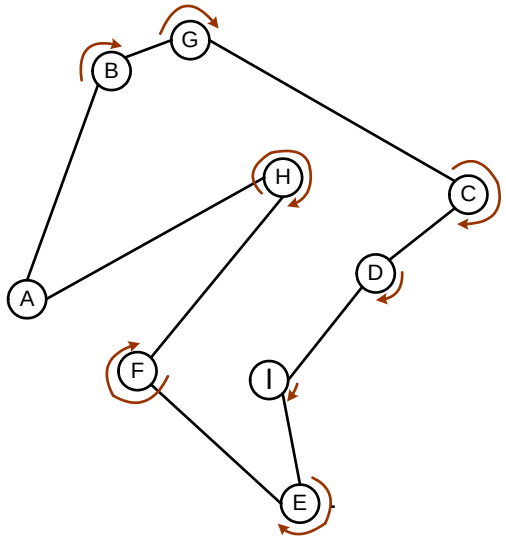
1. Eerst moet je het probleem begrijpen
2. Stel vervolgens een plan op
3. Voer het plan uit
4. Kijk terug naar je werk, wat kan je eraan verbeteren?

Heuristische algoritmen gebruiken probleemkennis om de oplossingsruimte te doorzoeken en zo de explosie hiervan te beperken. Ze proberen het zoeken te richten naar de meest belovende regio. Het nadeel van deze heuristische methodes is dat ze vaak degenereren naar een exhaustieve zoekfunctie (Silver, 2002).

Vanaf het begin van de jaren '70 werd het duidelijk dat het praktisch onmogelijk is om een optimale oplossing te vinden voor problemen zoals VRP en TSP. De rol van heuristieken bij het oplossen van combinatorische problemen uit de praktijk werd steeds belangrijker (Gendreau, 2002).

We kunnen dit verduidelijken met het eerder genoemde probleem van de *Travelling Salesman*. In het uitgewerkte voorbeeld moet een handelsreiziger negen (A tot I) steden bezoeken, vertrekkende en eindigend in stad A. In meer realistischer problemen zal het aantal steden echter veel hoger zijn. In figuur 2.4 wordt de volgende heuristische strategie voorgesteld:

- Vertrek vanuit stad A naar een willekeurige stad, bijvoorbeeld stad B
- Voor elke stad B tot H:
  - Kijk met je aangezicht naar de stad waarvan je komt
  - Begin nu naar rechts te draaien
  - Ga naar de eerste stad die je tegenkomt



Figuur 2.4: TSP opgelost met een heuristische strategie

We kunnen uit de grafiek concluderen dat deze snel gevonden oplossing niet de optimale zal zijn. Zoals eerder al aangegeven had het veel langer geduurd om de allerbeste oplossing voor dit probleem te vinden. Er bestaan echter methodes die tot een kwalitatief betere oplossing leiden dan klassieke heuristieken, terwijl ze veel minder rekentijd nodig hebben dan exacte methodes, namelijk metaheuristieken (Silver, 2002).

2.3 Metaheuristiek

2.3.1 Definitie

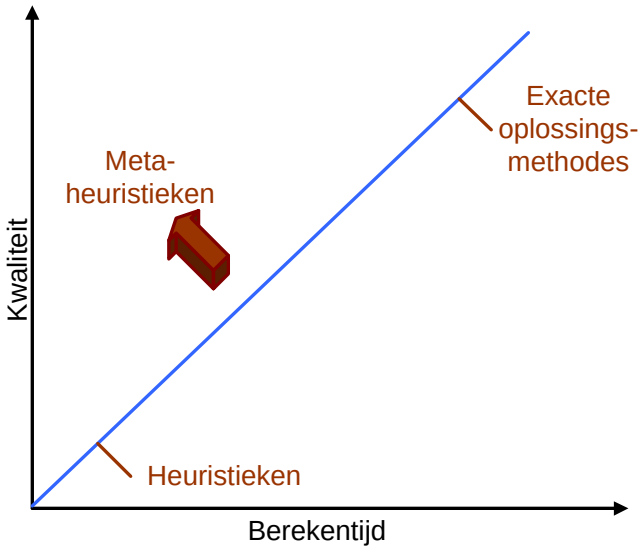
Zoals de naam al laat vermoeden houden metaheuristieken (MH) verband met de klassieke heuristieken. Het woord metaheuristiek bestaat uit twee delen: 'meta' en 'heuristiek'. De term *meta* betekent 'verder', 'op een hoger niveau'. Metaheuristieken zijn geen heuristieken, maar raamwerken. Het zijn richtlijnen om een heuristiek te maken. Omdat ze specifiek op een bepaald probleem worden toegepast, zal het resulterende algoritme verschillen van probleem tot probleem. Metaheuristieken zijn toepasbaar op vrijwel alle combinatorische optimalisatieproblemen (Sörensen).

We vinden volgende definities terug in de literatuur:

> A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem (MetaheuristicsNetwork, 2004).

> A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions (Osman en Laporte, 1996), zoals aangehaald in (Blum en Roli, 2003).

In tegenstelling tot een exacte oplossingsmethode zoekt een metaheuristiek niet dé allerbeste oplossing, maar probeert op een snelle manier tot een goede oplossing te komen. Deze unieke balans tussen kwaliteit en snelheid zorgt ervoor dat metaheuristieken steeds meer worden gebruikt voor het oplossen van diverse optimalisatieproblemen. Figuur 2.5 geeft de verschillen weer tussen exacte oplossingsmethodes, heuristieken en metaheuristieken.



Figuur 2.5: Vergelijking van verschillende optimalisatietechnieken

2.3.2 Globale indeling

Er zijn een aantal verschillende metaheuristieken. We kunnen ze in drie grote categoriën onderverdelen (Blum en Roli, 2003), zoals ook te zien is in figuur 2.1:

1. local search gebaseerde metaheuristieken
2. populatiegebaseerde metaheuristieken
3. hybride metaheuristieken

In de hierop volgende secties worden deze drie groepen meer in detail besproken.

Local Search gebaseerde MH

Bij local search MH gaan we van de ene naar de andere oplossing door kleine wijzigingen aan te brengen (de zogenaamde *moves*). Enkele voorbeelden van moves bij het handelsreizigersprobleem zijn:

- een stad op een andere positie zetten (zodat ze bijvoorbeeld later of vroeger wordt bezocht)
- de volgorde van twee steden omdraaien
- de volgorde van drie steden wisselen

- ...

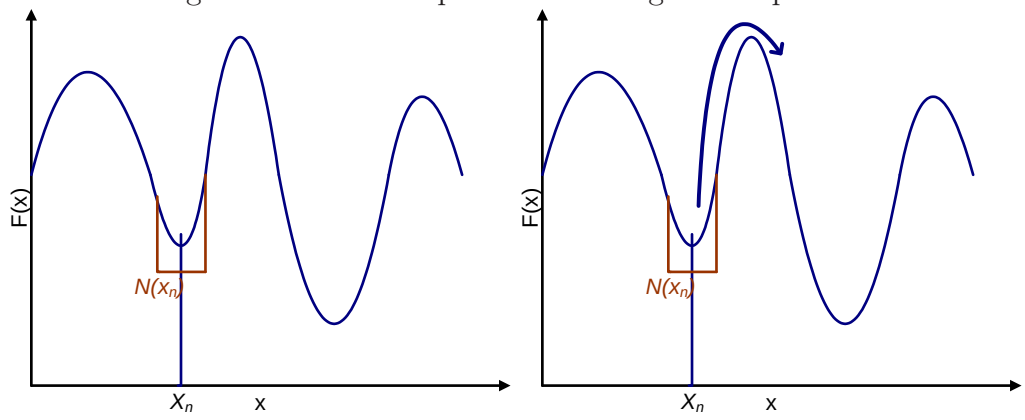
De *neighbourhood* $N(x_1)$ is de verzameling van alle oplossingen die we vanuit een huidige oplossing (x_1) kunnen bereiken door middel van één move (Focacci et al., 2001). De vraag is natuurlijk welke oplossing uit de neighbourhood we als volgende oplossing gaan aanvaarden, $x_2 \in N(x_1)$. Deze beslissing wordt bepaald door de *strategie* die we volgen. Twee voorbeelden van mogelijke strategieën zijn:

- steepest descent: hierbij aanvaarden we de beste oplossing in de neighbourhood.
- mildest descent: we genereren de neighbourhood maar stukje voor stukje. Vanaf het moment dat we een betere oplossing tegenkomen, dan aanvaarden we die (Silver, 2002).

Het concept van local search wordt nog verder geïllustreerd in sectie 2.4.4 over tabu search.

Een vaak voorkomend probleem is dat een strategie blijft vastzitten in een lokaal optimum. Een lokaal optimum is een oplossing waarbij onze strategie geen betere oplossing meer kan vinden, terwijl er nog wel een betere bestaat. Op de linkerhelft van figuur 2.6 is dit grafisch weergegeven. Vanuit punt x zou het kunnen dat de strategie niet verder kijkt dan de stijging rond x en dus niet weet dat er nog een lager punt is (Blum en Roli, 2003). De kracht van metaheuristieken is dat zij kunnen ontsnappen uit zo'n lokaal optimum en daardoor meer kans maken om het globale optimum, de allerbeste oplossing, te bereiken, zoals is voorgesteld in figuur 2.6. Het maken van een move wordt herhaald totdat een bepaald stop-criterium bereikt is.

Figuur 2.6: Lokaal optimum versus globaal optimum



Enkele bekende local search metaheuristieken zijn tabu search, simulated annealing, iterated local search. Deze worden uitgebreid besproken verder in dit hoofdstuk (secties 2.4.1, 2.4.2 en 2.4.4).

Populatiegebaseerde MH

Populatiegebaseerde MH hebben hun wortels in de genetische biologie en nemen dan ook een deel van deze nomenclatuur over. Ze behandelen steeds een aantal oplossingen samen, deze vormen een *populatie*. Om tot een volgende oplossing te komen, combineren ze bestaande oplossingen, waarbij hopelijk de beste eigenschappen van de 'ouders' behouden blijven. Het combineren gebeurt tot een bepaald stop-criterium bereikt is. Kenmerkend voor dit soort metaheuristieken is dat zij op een vrij willekeurige manier de oplossingsruimte verkennen, in tegenstelling tot local search strategieën die meer gericht verkennen (Dréo et al., 2003).

Bekende voorbeelden van populatiegebaseerde metaheuristieken zijn evolutionaire algoritmen, ant colony optimization,... Evolutionaire algoritmen worden verder besproken in sectie 2.4.3.

Hybride MH

Als derde categorie kunnen we de *hybride* metaheuristieken beschouwen. De laatste tijd is er een toenemende interesse in de literatuur vast te stellen voor hybride MH. Deze technieken zijn een combinatie van twee soorten metaheuristieken.

We onderscheiden drie types van hybride MH (Blum en Roli, 2003):

- MH die componenten van een andere MH opnemen
- coöperatieve zoeksystemen, algoritmen die info uitwisselen
- integratie van methodes

Zo zijn er bijvoorbeeld de memetische algoritmen (Corne et al., 1999). Deze zijn een combinatie van genetische en local search algoritmen. Meestal manifesteert deze combinatie zich als een evolutionair algoritme dat een local search strategie gebruikt in plaats van een mutatie-move (Prins, 2004). Bij een evolutionair algoritme wordt de oplossingsruimte vrij willekeurig doorzocht. De 'local search-mutatie' zorgt voor een meer gerichte zoektocht door de gigantische oplossingsruimte (Rodrigues en Ferreira, 2001).

Nummer	Voorwerp	Gewicht (kg)	Winstbijdrage
1	Tent	3	9
2	Kleren	2	7
3	Sandalen	1.2	2
4	Boek	0.5	1
5	Kookgerei	1.1	9
6	Geld	0.1	10
7	Wandelstokken	0.6	5
8	Stafkaarten	0.4	8
9	Slaapzak	0.75	5
10	Overlevingsgids	0.9	10
11	matje	1.2	4
12	snorkel	0.3	1
13	drinkbus	1.5	4
14	hamer	2.5	2
15	stoel	2.1	1

Tabel 2.1: De voorwerpen die de bergbeklimmer kan meenemen, samen met hun gewicht en winstbijdrage

2.3.3 Voorstelling van een oplossing

Om een probleem te kunnen optimaliseren is het noodzakelijk om het op een gestructureerde manier voor te stellen. Het is interessant om een deel van de structuur van een oplossing te behouden. Er zijn enkele fundamentele manieren om een optimalisatieprobleem voor te stellen, bijvoorbeeld binaire voorstellingswijze, permutatievoorstelling...

Bij het eerder besproken klassieke knapsack-probleem is een *binaire voorstellingswijze* aanbevolen. Dit kan het best verduidelijkt worden met behulp van een voorbeeld.

> Een bergbeklimmer wil een veertiendaagse trektocht maken door de Alpen. Zijn rugzak heeft een capaciteit van 12kg. Welke voorwerpen moet kan hij best meenemen, gegeven het gewicht en de winst van elk van de voorwerpen (tabel 2.1)?
>
> Dit probleem kunnen we voorstellen door een reeks nullen een enen (vandaar de naam binair). Een 1 op de n-de plaats in de rij betekent dat element n wordt meegenomen. Bijvoorbeeld:

- In het geval dat er geen enkel object wordt meegenomen:

  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

- In het geval dat enkel object nummer 4 en 10 worden meegenomen:

  0 0 0 1 0 0 0 0 0 1 0 0 0 0 0

Deze voorstellingswijze is duidelijk zeer eenvoudig en toch gestructureerd. Helaas zijn niet alle problemen even geschikt om binair te worden voorgesteld. In het geval van het Travelling Salesman Problem is een *permutatie-voorstelling* aan te raden. Hieronder wordt dit verduidelijkt met een voorbeeld.

> Een verkoper van vrachtwagenonderdelen moet dagelijks zes garages (A, B, C, D, E en F) bezoeken om ze in te lichten over de nieuwste ontwikkelingen op de markt. Gegeven de posities van de steden, wat is de kortste route voor de verkoper?
>
> Een oplossing kan worden voorgesteld door een rij geordende letters of getallen. De volgorde van deze letters bepaalt de volgorde waarin de verkoper de garages bezoekt. Bijvoorbeeld:
>
> B A F E C D
>
> Hierbij begint de verkoper in garage B, om vervolgens langs A, F, E, C te gaan en te eindigen bij garage D.

Er zijn nog een heleboel andere voorstellingswijzen, vb. een matrixvoorstelling die binaire voorstellingen combineert. Vaak wordt een representatie van de oplossing individueel aangepast aan het probleem. In het uitgewerkt voorbeeld dat we gaan bekijken in hoofdstuk 4 zal dit ook het geval zijn.

2.4 Enkele bekende Metaheuristieken

2.4.1 Iterated Local Search

Een metaheuristiek is bij voorkeur zeer simpel, zowel conceptueel als praktisch, terwijl hij toch efficiënt blijft. Iterated local search (ILS) voldoet aan deze eigenschappen (Lourenco et al., 2001). De essentie van ILS is dat het op een iteratieve manier oplossingen genereert door een onderliggende heuristische methode. Dit leidt tot betere oplossingen dan herhaalde willekeurige runs van die heuristiek. Volgens

(MetaheuristicsNetwork, 2004) zijn er twee kernpunten die van een algoritme een ILS maken:

- er wordt een enkele keten gevolgd (dit geldt zeker niet voor populatiegebaseerde algoritmen)
- de zoektocht naar betere oplossingen gebeurt in een verkleinde zoekruimte die gedefinieerd wordt door de output van een black-box heuristiek.

Ondanks zijn zeer simpele aard heeft het algoritme een lange geschiedenis achter de rug. Ondermeer daardoor heeft het een resem verschillende benamingen gekregen, zoals iterated descent, large-step Markov-chains, iterated Lin-Kernighan en chained local optimization (Lourenco et al., 2001).

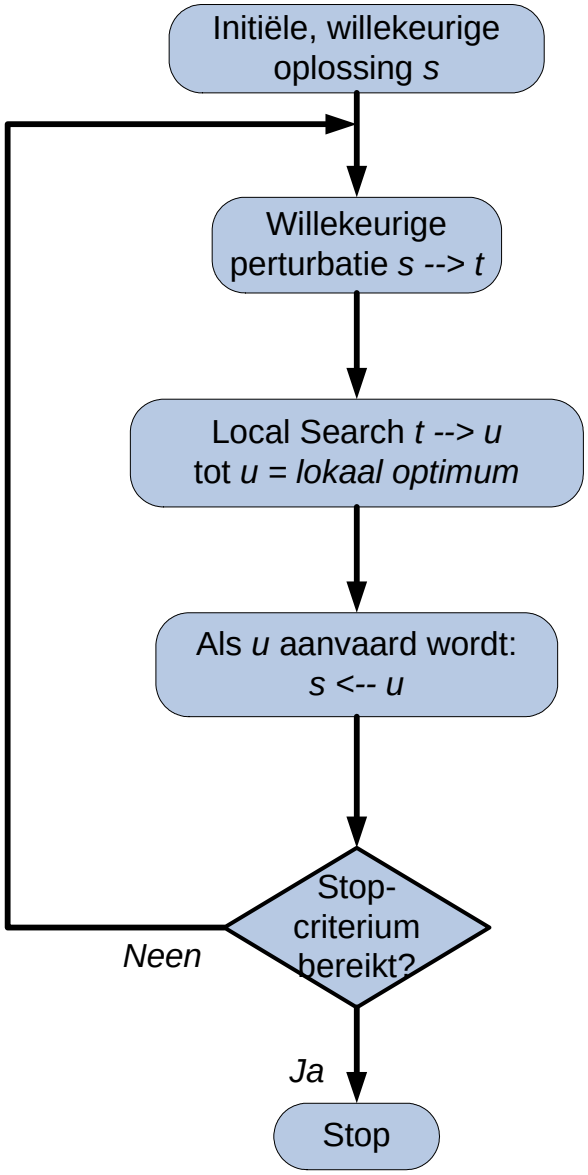
De werking is kort weergegeven in figuur 2.7. We vertrekken van een willekeurige oplossing. Eerst passen we daarop een local search strategie toe zoals die is beschreven in sectie 2.3.2. Dit doen we tot we een lokaal optimum bereiken. Dan volgt er een 'perturbatie', een sprong naar een andere plaats in de zoekruimte. Van daaruit gaat de local search strategie verder. Dit proces wordt herhaald tot een vooraf bepaald stop-criterium is bereikt (Besten et al., 2001).

De toegepaste perturbatie moet groot genoeg zijn, dit om te vermijden dat we meteen terugkeren naar het vorige locale optimum. Langs de andere kant mag de sprong ook niet te groot zijn (Lourenco et al., 2001).

2.4.2 Simulated Annealing

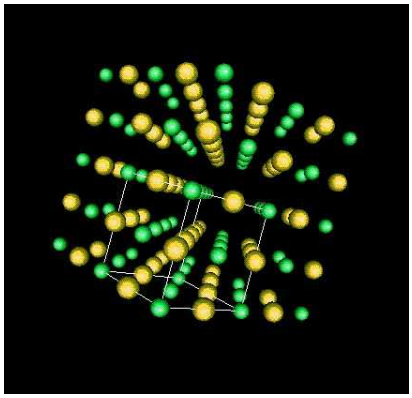
1983 was een belangrijk jaar op het gebied van combinatorische optimalisatie. Kirchpatrick, Gelatt en Vecchi stelden een nieuwe heuristische methode voor, nl. Simulated Annealing (Gendreau, 2002). Opmerkelijk aan deze techniek is dat men heeft aangetoond dat zij steeds converteert naar een optimale oplossing na een oneindig aantal moves. Deze ontdekking wekte de interesse van de onderzoekswereld. In de jaren die hierop volgden zijn er nog een heleboel verschillende methodes ontwikkeld, deze waren ook meestal gebaseerd op natuurkundige processen (Gendreau, 2002).

Simulated Annealing is gebaseerd op een fysisch proces, nl. het afkoelen van een kristallijne stof. Bij een kristallijne stof zijn alle atomen gerangschikt in een kristalrooster (zie figuur 2.8). Wanneer men de temperatuur doet stijgen, krijgen de atomen meer energie en gaan ze sneller bewegen. Hierdoor zitten ze niet meer vast



Figuur 2.7: Iterated Local Descent algoritme, gebaseerd op (Lourenco et al., 2001)

in het rooster. Deze toestand van wanorde kan men vergelijken met de begintoestand van een optimalisatieprobleem. De structuur van het kristalrooster stelt de oplossing van het optimalisatieprobleem voor (Dréo et al., 2003).



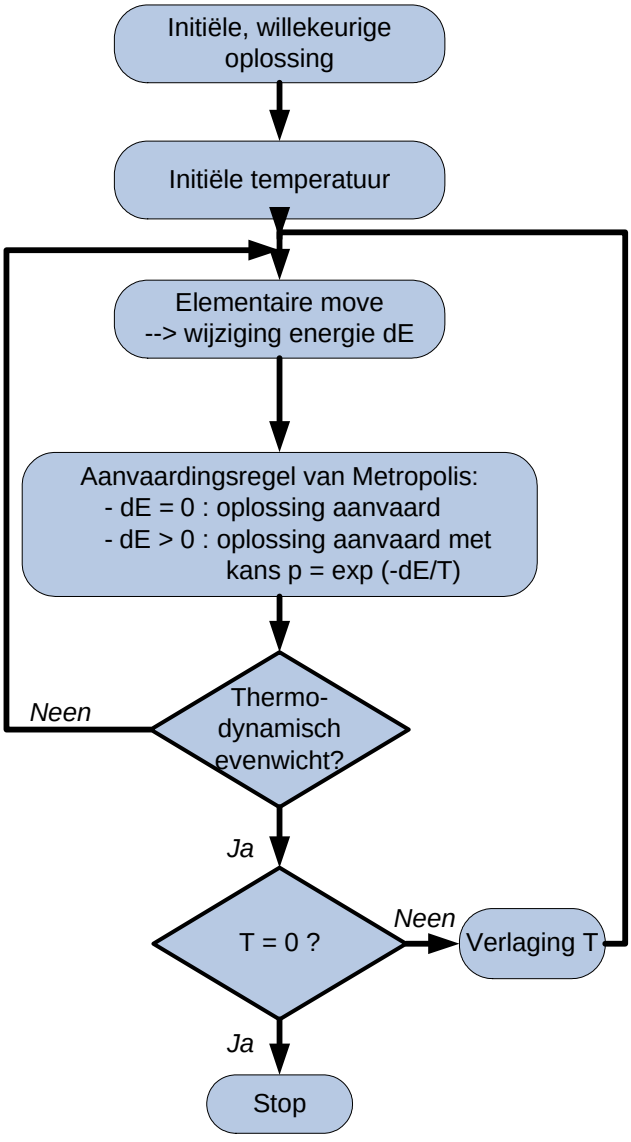
Figuur 2.8: Na^+ en Cl^- in het kristalrooster van keukenzout (RUN, 2004)

Bij het afkoelen van zo'n kristallijne stof kunnen er imperfectheden optreden. Dit zijn lokale optima, want elke wijziging zorgt voor een energie-verhoging. De bedoeling is echter om een globaal optimum te bereiken waarbij de energietoestand zo laag mogelijk is. De objectieffunctie komt in dit geval overeen met de mate waarin alle atomen op hun juiste plaats zitten in het kristalrooster ($\approx$ lage energietoestand). Een 'move' is een herschikking van de atomen (daling of stijging van de temperatuur) (Kirkpatrick et al., 1983).

Het algoritme is voorgesteld in figuur 2.9. We vertrekken van een willekeurige, initiële oplossing x, met een begintemperatuur T. We maken vervolgens een random move vanuit deze oplossing. De move wordt met zekerheid geaccepteerd als hij de oplossing verbetert (lagere energie). Wanneer de oplossing slechter wordt passen we de beslissingsregel van Metropolis toe (Kirkpatrick et al., 1983): de oplossing wordt aanvaard met een kans $p = e^{-\frac{\Delta E}{T}}$.

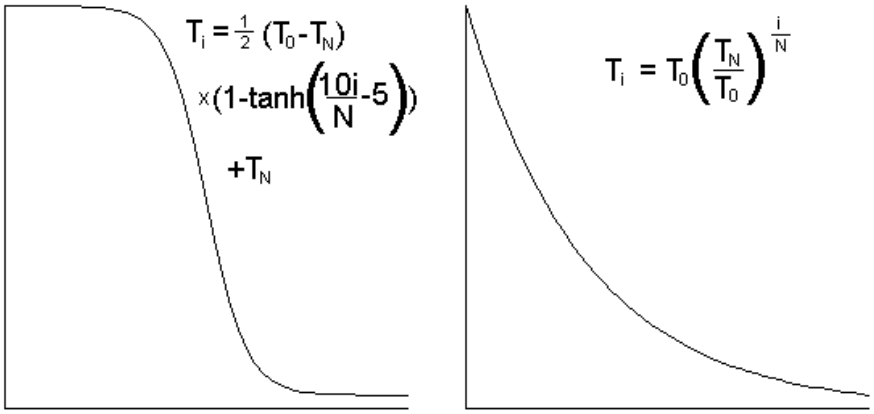
De kans p om de oplossing te aanvaarden wordt beïnvloed door 2 variabelen, de temperatuur T en de stijging van de objectieffunctie ΔE (Dréo et al., 2003):

- Hogere $T \rightarrow$ Hogere kans p (en omgekeerd: $\downarrow T \rightarrow \downarrow p$):
  $\uparrow p = e^{-\frac{\Delta E}{T\uparrow}\downarrow} \uparrow$
- Hogere $\Delta E \rightarrow$ Lagere kans p (en omgekeerd: $\downarrow T \rightarrow \uparrow p$):
  $\downarrow p = e^{-\frac{\Delta E\uparrow}{T}\uparrow} \downarrow$



Figuur 2.9: Simulated Annealing (Dréo et al., 2003)

Een belangrijke observatie is dat men zelf het aantal aanvaarde moves kan controleren door T aan te passen. Dit gebeurt door de invoering van een *afkoelingsschema*. Enkele mogelijke afkoelingsschema's zijn weergegeven in figuur 2.10.



Figuur 2.10: Mogelijke afkoelingsschema's (Luke, 2004)

Om de werking van dit algoritme beter te begrijpen is het interessant om simulated annealing toe te passen op het eerder aangehaalde voorbeeld van het knapsackprobleem.

We gaan verder met het knapsack probleem dat gegeven is in tabel 2.1. Stel dat we op een gegeven ogenblik over volgende gegevens beschikken:

- $T = 20$
- de huidige oplossing is:

  0 1 1 1 0 1 0 1 1 1 1 0 1 1 1

- het gewicht van deze oplossing is $8.2kg$. De oplossing is geldig ($8.2 \leq 12$)
- de winst van deze oplossing bedraagt 54.

Een mogelijke move die kan worden toegepast is het omdraaien van één bit. Stel bijvoorbeeld dat de volgende move het omdraaien van de eerste bit is. Dan krijgen we het volgende:

- de oplossing is:

  1 1 1 1 0 1 0 1 1 1 1 0 1 1 1

- het gewicht van deze oplossing is $11.2kg$. De oplossing is geldig $(11.2 \leq 12)$
- de winst van deze oplossing bedraagt 63.

Deze move wordt zeker aanvaard omdat hij een daling van de energie teweeg brengt ($\Delta winst = \Delta E = 63 - 54 = 9$). De temperatuur wordt dus ook verlaagd, $T = 19$ en we gaan verder met de volgende move. Stel dat de volgende move het willekeurig omdraaien is van bit 2, dan krijgen we volgende oplossing:

- de oplossing is:

  1 **0** 1 1 0 1 0 1 1 1 1 0 1 1 1

- het gewicht van deze oplossing is $9.2kg$. De oplossing is geldig $(9.2 \leq 12)$
- de winst van deze oplossing bedraagt 56.

Aangezien deze move een stijging van de energie teweegbrengt, wordt de beslissingsregel van Metropolis toegepast. De kans p om deze move te aanvaarden is:

$$p = e^{-\frac{\Delta E}{T}} = e^{(-\frac{9}{19})} = 0.623$$

Hiervoor trekken we een willekeurig getal u tussen 0 en 1. Als u groter is dan p dan zullen we de move verwerpen, en anders zullen we hem aanvaarden.

Over het proces van Simulated Annealing is reeds veel theoretisch onderzoek verricht. Zo is er ondermeer bewezen dat het algoritme na een oneindig aantal moves steeds tot de optimale oplossing komt (Blum en Roli, 2003). We moeten ons echter wel rekening geven van het feit dat een dergelijk bewijs niet aantoont dat het een goed algoritme is. Het is in de praktijk belangrijk om snel een kwalitatief goede oplossing te vinden, en niet na een oneindig lange tijd de beste oplossing te vinden. Dit brengt ons weer naar het belang van metaheuristieken, zoals dit is besproken in sectie 2.3.

2.4.3 Genetische Algoritmen

De geschiedenis van genetische algoritmen (GAs) begint in 1975, toen Holland het boek "Adaptation in Natural and Artificial Systems"(Holland, 1975) schreef. Doordat dit boek zo'n enorme wetenschappelijke interesse heeft teweeggebracht, wordt er wel eens naar verwezen als "The Bible of the GA Community"(Pirlot, 1996).

Figuur 2.11: John Holland (Vidal, 2003)

In tegenstelling tot de hiervoor besproken local search metaheuristieken, zijn GAs *populatiegebaseerde* MH (Pirlot, 1996). Ze behoren tot de klasse van 'evolutionaire algoritmen'. Deze evolutionaire algoritmen bevatten naast genetische algoritmen ook 'genetische programmering' en 'evolutionary strategies' (Digalakis en Margaritis, 2001).

De techniek is sterk geïnspireerd op het biologische voortplantingsproces en neemt dan ook heel wat termen uit de biologie over. Hieronder verklaren we de belangrijkste:

Populatie Een verzameling van individuen.

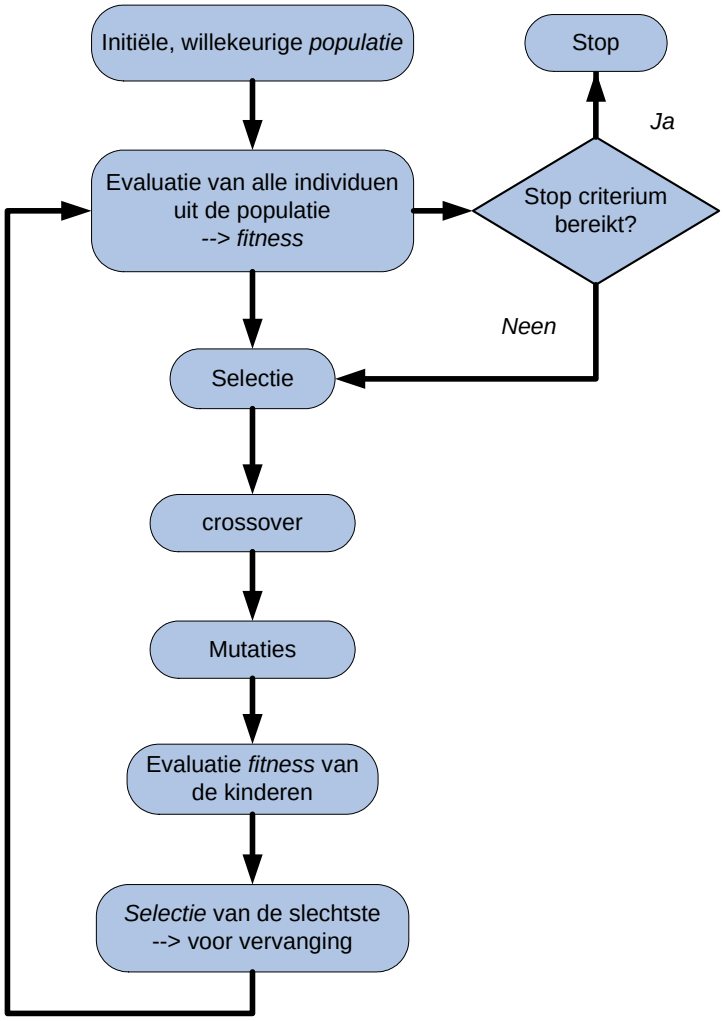
Individuen Individuele oplossingen, ook wel een **chromosomen** genoemd (Pirlot, 1996).

Fitness Geeft de waarde van de doelfunctie weer voor een individu. Oplossingen met een hogere fitness zijn beter aangepast aan de omgeving, zij hebben meer kans om te overleven en hun genen door te geven: "Survival of the fittest"(Blum en Roli, 2003)

Selectie Het selecteren van individuen uit een populatie die in aanmerking komen voor reproductie. Individuen met een hogere fitness zullen meer kans hebben om geselecteerd te worden (Dréo et al., 2003).

Crossover Het combineren van individuen (*parents*), om zo hopelijk tot een betere individuen (*children*) te komen. Dit is het *reproductieproces* (Dréo et al., 2003).

Mutatie Een kleine verandering in het genetisch materiaal. Dit is in veel gevallen negatief, maar soms kan het tot een beter individu leiden. Naar dit proces refereert men soms als *zelfadaptatie* (Blum en Roli, 2003).



Figuur 2.12: Genetisch algoritme (Dréo et al., 2003; Towsey et al., 2001)

Het algoritme is voorgesteld in figuur 2.12. We vertrekken van een initiële populatie van n oplossingen. Vervolgens berekenen we voor elk van deze individuen de *fitness* waarde. Op basis hiervan kunnen we de beste oplossingen *selecteren* met behulp van binaire toernooi selectie of roulette wheel selectie.

Bij *roulette wheel selectie*, selecteren we een oplossing met een kans die proportioneel is aan de fitness-waarde (Dréo et al., 2003). Stel dat we drie individuen in de populatie hebben, met respectievelijke fitness waarden: 3, 5 en 6. We gaan deze drie oplossingen selecteren met als kans: $\frac{3}{14}$, $\frac{5}{14}$ en $\frac{6}{14}$.

In het geval van *binaire toernooi selectie* selecteren we willekeurig twee individuen

en houden we enkel de beste bij. Een belangrijk voordeel hiervan is dat er geen risico is om de allerbeste uit de populatie te gooien (Prins, 2004).

Net zoals het in de natuur ook het geval is, zal bij GAs het meest aangepaste individu overleven en zijn 'genen' doorgeven, daarom gaan we de geselecteerde (goeie) oplossingen combineren. Een volgende stap in het algoritme is het kruisen van het genetisch materiaal van de geselecteerde individuen (= ouders), via het *crossover* proces. Er zijn verschillende vormen van crossover mogelijk, bijvoorbeeld 1-point crossover, 2-point crossover en OX crossover.

Bij *1-point crossover* kiest men 1 kruisingspunt en verwisselen we daarrond het genetisch materiaal. Bijvoorbeeld:

$$\begin{array}{cccc|cc} 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{array}$$

$$\Downarrow$$

$$\begin{array}{cccccc} 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{array}$$

2-point crossover maakt gebruik van hetzelfde principe, maar dan wel met 2 kruisingspunten. Bijvoorbeeld:

$$\begin{array}{cc|cc|cc} 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{array}$$

$$\Downarrow$$

$$\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{array}$$

Order crossover (OX) wordt toegepast bij permutatievoorstellingen. Het werkt als volgt:

- We trekken twee kruisingslijnen, net zoals bij 2-point crossover.
- We behouden het middelste stuk en schrijven dit dus gewoon over van de ouders.
- Voor elk kind:
  - begin na de tweede streep met het invullen van wat er bij de andere ouder staat na de tweede streep.

- controleer voor het invullen van een cijfer of het niet reeds is gebruikt. Dubbele getallen mogen niet voorkomen, in dit geval slagen we het getal over en gaan we naar het volgende.
- als we achteraan zijn gekomen, gaan we vooraan verder.

Bijvoorbeeld:

$$\begin{array}{cc|cc|cc} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 3 & 2 & 1 & 6 & 5 \end{array}$$

$$\Downarrow$$

$$\begin{array}{cccccc} 2 & 1 & 3 & 4 & 6 & 5 \\ 3 & 4 & 2 & 1 & 5 & 6 \end{array}$$

Het resultaat van deze crossover operatie zijn de *kinderen*. In een volgende stap kunnen er willekeurige *mutaties* optreden. Dit kan bijvoorbeeld het omdraaien van een bit zijn. Daarna evalueren we de fitness van de kinderen en selecteren we de slechtste individuen uit de populatie, dit kan weerom gebeuren met de methodes die hierboven beschreven staan. Dit proces wordt herhaald totdat een bepaald stop-criterium voldaan is.

2.4.4 Tabu Search

Figuur 2.13: Fred Glover (Glover, 2004)

Tabu Search (TS) is voor het eerst gepresenteerd door Fred Glover eind jaren '70 (Dréo et al., 2003). Zijn twee originele artikels (Glover, 1990)(Glover, 1989) bevatten veel opvattingen die nog steeds gelden voor tabu search (Dréo et al., 2003). Het heeft lang geduurd vooraleer al zijn opmerkingen gebruikt werden in de wetenschap. Vooral de aspiratieconditie en het principe van een tabulijst (zie verder in deze

sectie) werden al vanaf de jaren '90 gebruikt door wetenschappers die tabu search toepasten. Het gebruik van andere structuren liet langer op zich wachten (Glover, 1998). Op dit moment wordt er nog steeds veel onderzoek verricht naar TS en blijft de techniek evolueren en verbeteren (Perry Gray et al., 2002).

Zoals eerder al bleek is Tabu Search een local search algoritme, met een bijzondere eigenschap, het bevat namelijk geheugenstructuren. Glover beschrijft het concept tabu search als "a meta-heuristic superimposed on another heuristic"(Perry Gray et al., 2002). Het is dus een metaheuristiek die een local search heuristiek helpt om een zoekruimte efficiënt te verkennen. Het basisidee van deze metaheuristiek is vermijden dat het zoekproces vast komt te zitten in cirkels. Dit wordt ondermeer bereikt door bepaalde moves te verbieden, meer specifiek, diegene die iets doen wat juist gedaan is. Deze 'verboden' oplossingen zijn taboe, vandaar de naam *tabu search* (Laguna en Glover, 2004).

We kunnen een aantal verschillende geheugenstructuren onderscheiden. Fundamenteel kunnen we deze onderverdelen in korte- en lange termijn-geheugenstructuren.

Korte termijn-geheugenstructuren

Wellicht de belangrijkste geheugenstructuur is de *tabulijst*. Deze lijst bevat de laatst bezochte oplossingen (dit kunnen eventueel ook moves of attributen van oplossingen zijn) en heeft als doel te verhinderen dat we op onze stappen terugkeren (Silver, 2002). Een eigenschap van de tabulijst is de 'tabu tenure', de lengte van de lijst. Wanneer de tabulijst de k-laatste oplossingen bevat, dan zal de tenure k zijn. De moves die we niet mogen uitvoeren omdat de tabulijst ze verbiedt, noemen we 'tabu actief'.

Een tweede korte termijn geheugenstructuur is het *aspiratieniveau*. Hierbij houden we steeds de allerbeste oplossing bij. Wanneer we op een gegeven ogenblik tijdens het zoekproces een oplossing tegenkomen die beter is dan het aspiratieniveau, dan gaan we die zeker aanvaarden (Blum en Roli, 2003).

Ten slotte bestaan er ook nog de *kandidaat lijsten*. Deze lijsten bevatten bepaalde moves waarvan we op voorhand denken dat ze interessant zijn. De moves die een grote bijdrage leveren gaan we vervolgens meer laten voorkomen (Dréo et al., 2003).

Lange termijn-geheugenstructuren

De LT-geheugens zorgen ervoor dat TS twee belangrijke componenten bezit, nl. intensificatie en diversificatie. Tijdens de *intensificatie* focust de strategie zich op het verkennen van de neighbourhood en de elite oplossingen. Terwijl bij de *diversificatie* de nadruk ligt op het exploreren van niet verkende ruimten, wat leidt tot oplossingen die significant verschillen van diegene die ervoor gekend waren (Blum en Roli, 2003).

Er zijn vier fundamentele vormen van lange termijn geheugen. Een eerste type is het *frequentiegeheugen*. Hierbij gaan we bijhouden hoeveel keer een bepaald soort oplossing voorkomt. Dit kan bijvoorbeeld worden geïmplementeerd onder de vorm van een matrix die bijhoudt hoeveel keer elk attribuut voorkomt. Het kan nuttig zijn voor de zoekstrategie om af en toe te starten van een attribuut dat nog niet veel is voorgekomen (Dréo et al., 2003).

Een andere vorm van geheugen is het *elitegeheugen*. Dit is een lijst met heel goeie oplossingen die we al zijn tegengekomen. Ook hier kan het nuttig zijn om te vertrekken van een eliteoplossing als de zoekfunctie vast zit (Sörensen, 2002).

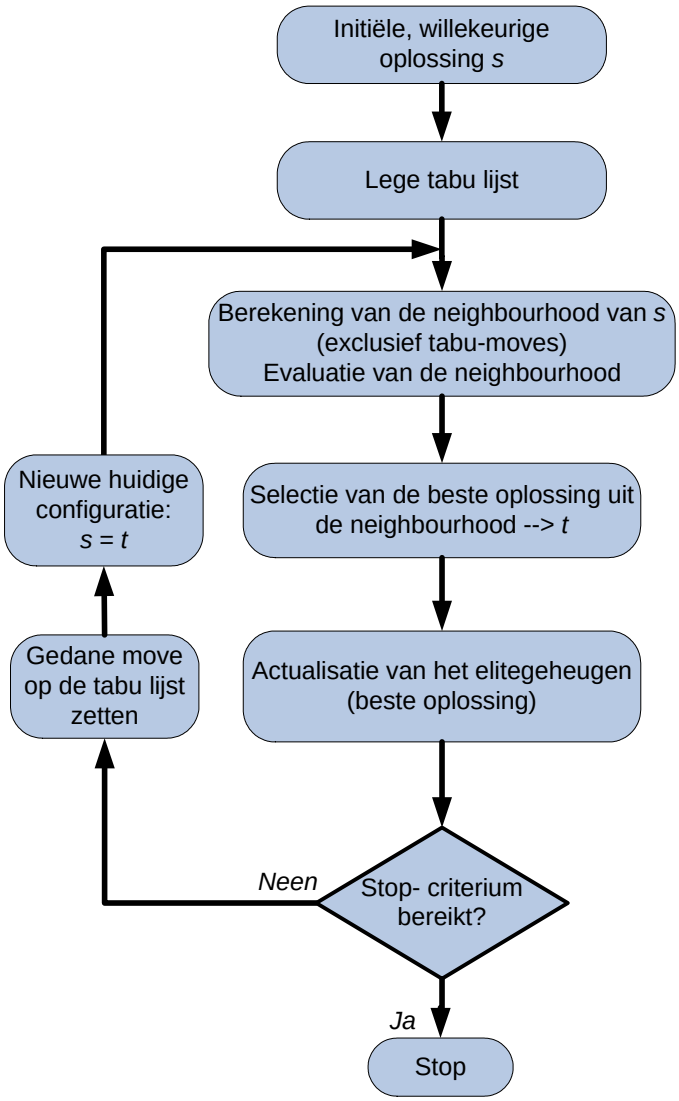
Vervolgens is er ook de *strategische oscillatie*. Dit wil zeggen dat we toelaten dat de oplossing tijdelijk niet geldig (ook wel 'infeasible' genoemd) wordt. Tenslotte hebben we ook de *reactive tabu search*. Dit is tabu search waarbij de tabu tenure kan worden aangepast in de loop van het zoekproces (Wikipedia, 2004).

De basiswerking van het algoritme is voorgesteld in figuur 2.14. We vertrekken van een initiële oplossing s en een lege tabulijst. Vervolgens gaan we de neighbourhood genereren. Bij een binaire oplossing kan dit bijvoorbeeld gebeuren door elke bit om te draaien. De moves die op de tabulijst staan gaan we echter niet uitvoeren. Hierop volgend selecteren we de beste oplossing uit de neighbourhood. Als het stop-criterium niet voldaan is gaan we de gedane move op de tabulijst zetten en vertrekkend van de gekozen oplossing de neighbourhood opnieuw genereren, enzoverder (Pirlot, 1996).

Om enkele van deze abstracte begrippen iets duidelijker te maken, illustreren we het tabu search algoritme aan de hand van het in knapsack probleem dat in tabel 2.1 is voorgesteld.

We gaan verder met een knapsack probleem dat gegeven is in tabel 2.1.
Stel dat we op een gegeven ogenblik over volgende gegevens beschikken:

- Tabulijst $= [5, 7, 8, 9]$, tenure $= 4$.



Figuur 2.14: Tabu Search Algoritme (Dréo et al., 2003)

- de huidige oplossing is:

  0 1 1 1 0 1 0 1 1 1 1 0 1 1 1

- het gewicht van deze oplossing is $8.2kg$. De oplossing is geldig ($8.2 \leq 12$)
- de winst van deze oplossing bedraagt 54.

Stel dat het algoritme als move alle bits één voor één gaat omdraaien, om zo de neighbourhood te bepalen. De neighbourhood is dan van de ordegrootte $O(n)$. We gaan dan voor alle 15 oplossingen uit de neighbourhood, het gewicht en hun winst bepalen.

De neighbourhood is:

1	1	1	1	0	1	0	1	1	1	1	0	1	1	1
0	**0**	1	1	0	1	0	1	1	1	1	0	1	1	1
0	1	**0**	1	0	1	0	1	1	1	1	0	1	1	1
0	1	1	**0**	0	1	0	1	1	1	1	0	1	1	1
0	1	1	1	**1**	1	0	1	1	1	1	0	1	1	1
0	1	1	1	0	**0**	0	1	1	1	1	0	1	1	1
0	1	1	1	0	1	**1**	1	1	1	1	0	1	1	1
0	1	1	1	0	1	0	**0**	1	1	1	0	1	1	1
0	1	1	1	0	1	0	1	**0**	1	1	0	1	1	1
0	1	1	1	0	1	0	1	1	**0**	1	0	1	1	1
0	1	1	1	0	1	0	1	1	1	**0**	0	1	1	1
0	1	1	1	0	1	0	1	1	1	1	**1**	1	1	1
0	1	1	1	0	1	0	1	1	1	1	0	**0**	1	1
0	1	1	1	0	1	0	1	1	1	1	0	1	**0**	1
0	1	1	1	0	1	0	1	1	1	1	0	1	1	**0**

We kiezen vervolgens de beste oplossing uit deze 15 (diegene met de hoogste winst + feasible) en controleren of deze niet op de tabulijst staat (zie rood). Stel bijvoorbeeld dat de beste oplossing werd verkregen door het omdraaien van bit 5, dan zouden we deze niet aanvaarden, maar de op één na beste oplossing kiezen. Dit is een zeer handig mechanisme om te vermijden dat we in cirkeltjes blijven vastzitten. Stel dat de beste oplossing die geen tabu is, oplossing 4 is. We nemen deze oplossing over als nieuwe oplossing, passen de tabulijst aan en we beginnen terug opnieuw.

2.5 Keuze van de juiste metaheuristiek

Omdat metaheuristieken richtlijnen zijn om een heuristiek te maken, moeten ze worden aangepast aan elk probleem. Bepaalde metaheuristieken zullen echter gemakkelijker aan te passen zijn aan verschillende problemen en sneller tot een goeie oplossing komen. Zo zijn er voor het vehicle routing problem een aantal zeer efficiënte tabu search-methodes geschreven en geïmplementeerd, maar bijna nog geen genetische algoritmen. Prins is de enige die er met een memetisch algoritme (combinatie van genetische algoritmen en local search) de tabu search programma's naar de troon steekt (Prins, 2004).

De keuze van de juiste metaheuristiek is niet altijd even simpel. Er zijn geen goeie bestaande theorieën die ons hierbij kunnen helpen (Dréo et al., 2003). Daarom is de keuze van een goede MH en de aanpassing van de bijhorende parameters vaak afhankelijk van de bekwaamheid en de ervaring van de gebruiker in plaats van de exacte toepassing van de voorgeschreven regels.

Het *No Free Lunch-theorema* zegt ons bovendien dat er geen methode is die voor alle problemen de beste oplossing geeft.

> *... all algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost functions. In particular, if algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist exactly as many other functions where B outperforms A* (NoFreeLunchTheorems, 2004).

Dit wil zeggen dat het niet mogelijk is dat er op een dag een metaheuristiek wordt ontwikkeld die de beste oplossingsmethode is voor alle problemen.

3 CAC: enkele projecten

3.1 Inleiding

In de vorige twee hoofdstukken hebben we gezien dat we het componeren van muziek als een optimaliseringsprobleem kunnen beschouwen. We hebben ook iets meer gezegd over een bepaald soort oplossingsmethodes voor optimaliseringsproblemen, namelijk metaheuristieken. In dit hoofdstuk gaan we de twee vorige koppelen en enkele concrete voorbeelden geven van Computer Assisted Composing-projecten die reeds zijn gerealiseerd. De besproken projecten zijn GenJam en SIEDP. We beginnen met een overzicht van enkele meer praktische dingen zoals het voorstellen van muziek.

3.2 Hoe gaat CAC in zijn werk?

3.2.1 Voorstelling van Muziek

Er zijn verschillende mogelijkheden om muziek voor te stellen met een computer. Veruit de bekendste is de 'Common Hierarchical Abstract Representation for Music', ofwel de *CHARM*-methode. Deze abstracte methode wordt voorgesteld in (Smail et al., 1990) en is de eerste stap naar een algemeen notationeel systeem voor muziek. Ze gaat ervan uit dat elke noot een *event* is, en dat deze drie eigenschappen heeft:

Pitch De toonhoogte.

Time Het tijdstip waarop de noot begin.

Duration Het aantal tijden dat de noot blijft voortduren.

Deze drie eigenschappen vormen de basiseigenschappen. Ze kunnen nog worden uitgebreid met o.a. timbre, intensiteit, amplitude,... (Wiggins et al., 1989). Een mogelijke voorstelling van een noot met behulp van het *CHARM*-systeem, wordt gegeven in figuur 3.1.

```
event(Identifier, Pitch, Time, Duration, Amplitude,
                Relatieve Amplitude,...)
                        ↓
        event(e01, (f,♯ ,4), 1/4, 1/2, 10/1, ...)
```

Figuur 3.1: De *CHARM*-voorstellingswijze (Wiggins et al., 1993b)

De *CHARM*-methode is niet gebonden aan een specifieke programmeertaal, en kan dus in principe gebruikt worden op elk platform (Reynolds, 1995), van ML (een functionele taal) tot Prolog (een logische programmeertaal).

Een belangrijk voordeel van deze voorstellingswijze is dat ze op een relatief eenvoudige wijze een grammatica-aanpak toelaat (Wiggins en Smail, 2000). Grammatica doet meteen denken aan een taal en we kunnen ons hierbij dan ook afvragen of muziek kan beschouwd worden als een taal. Daarvoor zou er een verband moeten bestaan tussen het vlak van de expressie (hoorbare muziek) en het vlak van de inhoud (semantische structuur). Dit verband kunnen we ook omschrijven als 'betekent'. Het onderzoek omtrent de muzikale code is niet volledig, aangezien er niet voldoende begrip is van de vlakken van expressie en inhoud. We kunnen echter met zekerheid stellen dat er een verschil is tussen gesproken taal en muziek. Langs de andere kant is er een sterke neiging om verbanden tussen de twee te zien, zoals bijvoorbeeld grammatica (Baroni et al., 1983).

Een muziekstuk is geen discrete, onafhankelijke verzameling van geordende noten. Het bestaat uit 'zinnen', meestal vier maten lang, die op hun beurt hiërarchisch geordend zijn. Een muzikale zin wordt gedefinieerd als: "een sequentie die een punt bereikt dat een gevoel van pauze of aankomst aangeeft"(Ponsford et al., 1997). We moeten met deze zinnen en structuren rekening houden wanneer we muziek willen voorstellen. De *CHARM*-methode laat dit toe door een *constituent* te definiëren die bestaat uit verschillende events (Wiggins et al., 1989). Zo'n constituent is dus een verzameling noten die samen een zin vormen.

3.2.2 De kwaliteit van muziek meten

Een van de moeilijkheden bij CAC-toepassingen is het bepalen van de doelfunctie. Wat is de kwaliteit van de muziek? Naar dit probleem wordt vaak verwezen als de 'fitness bottleneck' (Gartland-Jones, 2003). *Fitness* verwijst naar de benaming van de doelfunctie bij genetische algoritmen. *Bottleneck* duidt op de moeilijkheid om een doelfunctie op te berekenen en op te stellen. De bedoeling is dat we een soort van *score* kunnen berekenen of toekennen aan de hand waarvan we de muzikale kwaliteit van het fragment kunnen kwantificeren (Biles, 1994).

Er zijn twee fundamenteel verschillende manieren om de 'score' van een muziekstuk te bepalen:

- interactief
- via een doelfunctie

De eerste manier is **interactief**. Individuen gaan de muziek beluisteren en er een score aan toekennen. Een belangrijk nadeel hiervan is dat deze score subjectief is (Gartland-Jones, 2002). Ze zal bijvoorbeeld sterk afhangen van de gemoedstoestand en aandacht (Wiggings et al., 1998). Bovendien gaan de iteraties enorm traag omdat het stuk telkens helemaal beluisterd en beoordeeld moet worden.

Ten tweede kunnen we aan de hand van formele regels een **doelfunctie berekenen**. Hierbij zijn er geen mensen nodig om de totaalscore te berekenen. Op basis van enkele criteria berekent de computer zelf de score van een oplossing. Van deze criteria wordt meestal het gewogen gemiddelde berekend op basis van hun belang. De totaalscore van deze gemiddelden is onze te optimaliseren waarde (Schoenberger, 2004). Er dient wel te worden opgemerkt dat de kwaliteit van de vooropgezette doelfunctie de resultaten sterk zal beïnvloeden.

In de volgende secties volgen enkele voorbeelden van interactieve en geautomatiseerde doelfuncties.

We hebben in sectie 1.1 gezien dat een optimaal muziekstuk aan enkele criteria moet voldoen zodat het een goede harmonische, melodische en ritmische samenhang heeft (Kleeven en Philippi, 2002; Schell, 2002). De moeilijkheid is het bepalen van de juiste criteria om deze samenhang te meten. Er zijn vele studies gebeurd, maar er bestaat geen algemene doelfunctie.

De meeste studies zijn gebaseerd op meerstemmige muziek en maken gebruik

van criteria op basis van de juiste opeenvolging van akkoorden. De volgende maatstaven worden onder andere gebruikt bij het optimaliseren van meerstemmige muziek: de juiste akkoordenprogressie, het aantal gebruikte gronddrieklanken[1],... (Schoenberger, 2004; Bilotta, 2000; B.Pardo, 2002).

3.2.3 Programmeertalen

Er zijn een aantal programeertalen beschikbaar die speciaal ontworpen zijn om er CAC-programma's mee te schrijven. Enkele voorbeelden:

- Nyquist
- CSound
- OpenMusic

We zullen hieronder kort iets vertellen over deze drie programmeertalen.

Nyquist

Nyquist is een programmeertaal die speciaal ontwikkeld is voor algoritmisch componeren en geluidssynthese. Ze ondersteunt zowel high-level componeren als low-level signaalverwerking. Dit is vooral interessant voor componisten die op het microscopisch abstractieniveau willen werken (Dannenberg, 2005). Nyquist is geïmplementeerd bovenop XLisp (een uitbreiding van Lisp). We kunnen Nyquist zien als een bibliotheek met Lispfuncties die kunnen worden opgeroepen door functies in een programma.

Het grote voordeel van Nyquist is dat componisten de volledige functionaliteit van Lisp kunnen gebruiken. Deze taal wordt veel gebruikt in het onderzoek naar artificiële intelligentie. Anderzijds moeten componisten reeds een goede kennis hebben van Lisp vooraleer ze voordeel kunnen halen uit de functionaliteit (Miranda, 2001).

CSound

In sectie 1.4.3 is reeds kort iets gezegd over de geschiedenis van CSound. De taal CSound is volledig geschreven in C. Wanneer we werken in CSound hebben we

[1] Gronddrieklanken zijn de basisnoten van een toonladder (Geeurickx, 1985).

steeds twee bestanden nodig: een 'orchestra file' (.orc) en een 'score file' (.sco). Het eerste bestaat uit een verzameling van instrumenten die de computer vertellen hoe hij geluid moet synthetiseren. Het tweede vertelt de computer wanneer hij het moet synthetiseren (Anoniem, 2004). In bijlage B wordt een voorbeeld van een .orc en .sco bestand gegeven.

OpenMusic

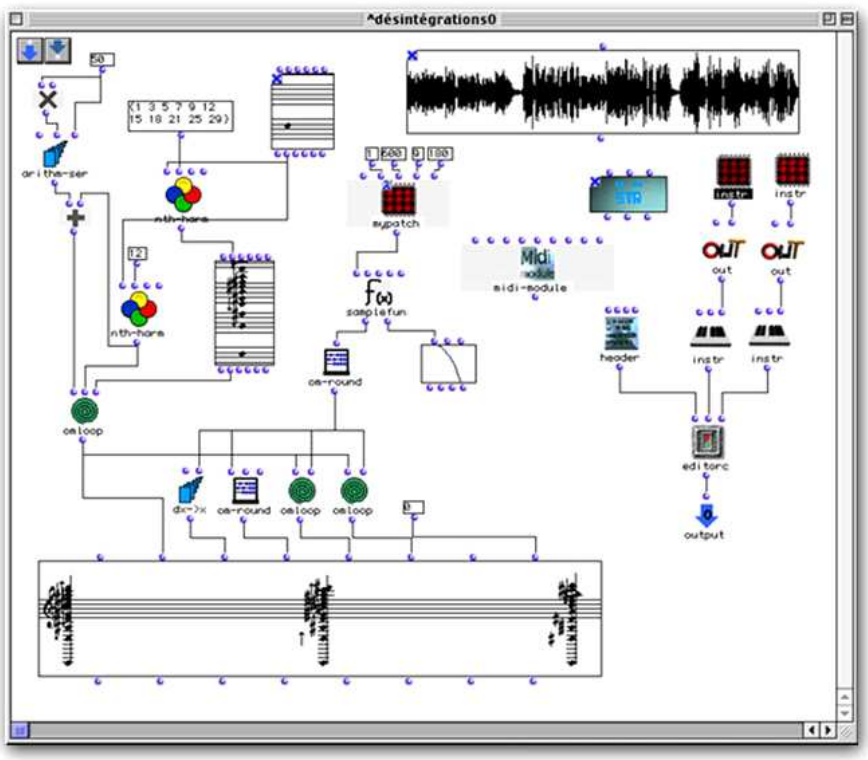
Gérard Assayag en Carlos Agon van het onderzoekscentrum IRCAM in Parijs, ontwikkelden OpenMusic (Truchet et al., 2003). Het is een systeem voor muziekprogrammering dat het midden houdt tussen een programmeertaal en algemene compositiesoftware. OpenMusic is een visuele interface voor de Lisp taal (CommonLisp of CLOS). De taal draait dus bovenop Lisp, net zoals Nyquist (Assayag et al., 1999).

OpenMusic is objectgeoriënteerd. Objecten worden gesymboliseerd door iconen die men over het scherm kan slepen. Door het slepen en plakken van deze iconen kunnen de meeste operaties gerealiseerd worden (IRCAM, 2005b). Een voorbeeld van de visuele programmeeromgeving wordt gegeven in figuur 3.2.

3.3 GenJam

3.3.1 Inleiding

GenJam is ontwikkeld door John A. Biles (Biles, 1994). Het programma imiteert een beginnende jazz musicus die solo's leert spelen. GenJam is, zoals de naam reeds aangeeft, gebaseerd op een interactief genetisch algoritme (zie sectie 2.4.3). De solo's die GenJam produceert komen voort uit toonladders die worden gesuggereerd op basis van de gekozen akkoord-progressies. De begeleiding van GenJam bestaat uit een standaard ritme en een menselijke mentor die real-time feedback geeft. Uit de feedback van de mentor wordt een fitness-waarde berekend. Daarna past GenJan verschillende genetische operatoren toe om zijn creatie te verbeteren. In wat hierop volgt bespreken we de verschillende onderdelen van GenJam iets meer in detail.

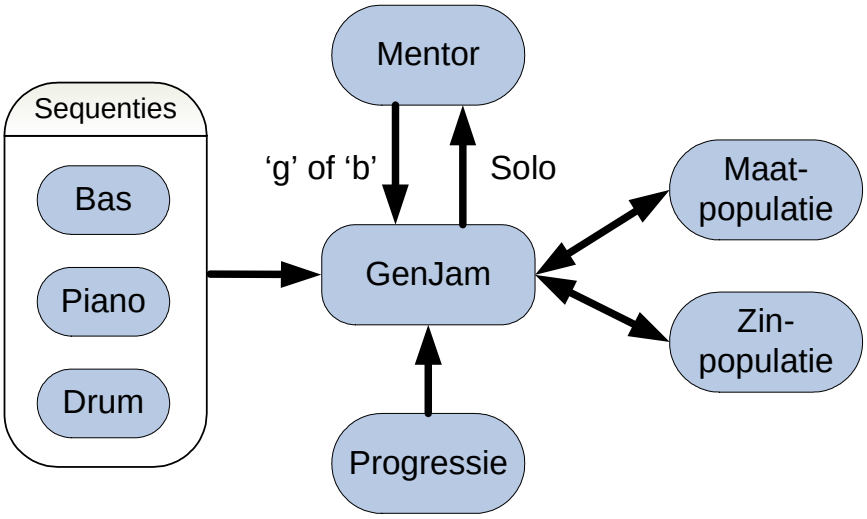


Figuur 3.2: De interface van OpenMusic (IRCAM, 2005a)

3.3.2 De werking van GenJam

De 'Genetic Jammer', of kortweg GenJam, is ontwikkeld in een Macintosh/Think C omgeving, bovenop een CMU MIDI-toolkit. Een visueel overzicht wordt gegeven in figuur 3.3. GenJam leest MIDI sequenties in voor piano, bas en drum. Deze instrumenten zullen zijn solo begeleiden. Verder leest GenJam een 'progressie'-bestand in (Biles, 2004). Dit bestand bevat informatie omtrent:

- ritmische stijl
- aantal solo-refreinen
- akkoordenprogressie



Figuur 3.3: De architectuur van GenJam (Biles, 2004)

Op basis hiervan wordt een solo gegenereerd in real-time. De mentor, een persoon die hiernaar luistert, zal de verschillende delen van de muziek quoteren met (Biles, 1996):

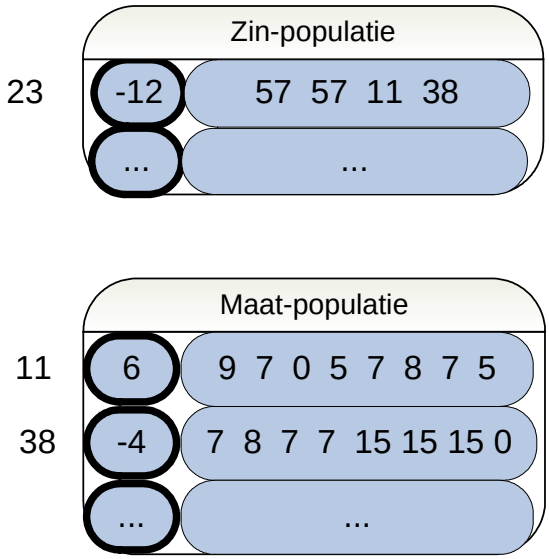
- g: goed
- b: slecht

Op basis hiervan genereert GenJam steeds een verbeterde versie het fragment.

3.3.3 Chromosoom-voorstelling

De manier waarop populaties worden voorgesteld is cruciaal voor het succes van een genetisch algoritme. GenJam gebruikt niet één, maar twee populaties: een van maten en een van 'zinnen'[2]. Een individu uit de maat-populatie bestaat uit een opeenvolging van MIDI-events, terwijl een individu uit de zin-populatie bestaat uit enkele maten. Het grote voordeel hiervan is dat de mogelijkheid bestaat dat een bepaalde maat te herhalen in een zin. Dit geeft de luisteraar vaak het gevoel dat er een thema inzit (Biles, 1994).

In figuur 3.4 wordt een voorbeeld gegeven van de gebruikte voorstelling. Individu 23 is een willekeurig individu van de zin-populatie. Zoals we zien bestaat deze populatie uit vier individuen uit de maat-populatie, namelijk 57, 57, 11 en 38. In functie van het gebruikte genetische algoritme zullen alle getallen in de figuur binair worden voorgesteld. Het geheel van alle individuen in de zin-populatie vormt het muziekstuk (Biles, 1996).



Figuur 3.4: Voorbeeldpopulaties (Biles, 1996)

[2]Muzikale zinnen zijn opeenvolgingen van meestal vier maten, in sectie 4.2 wordt hier meer over gezegd.

3.3.4 Fitness Bottleneck

Zoals we gezien hebben in sectie 1.2.3 is het zeer moeilijk om een 'fitness'- of doelfunctie te bepalen voor een muziekstuk. Biles heeft ervoor gekozen om geen algoritmische functie hiervoor te voorzien. In plaats daarvan bepaalt een menselijke criticus hoe goed het muziekfragment is (Biles, 2004).

Terwijl de mentor luistert naar de GenJam solo, kan hij ofwel een g (goed) ofwel een b (slecht) typen voor elke maat. Elke keer hij een g typt, verhoogt de fitness van de betreffende maat en zin. Bij een b verlaagt de fitness. Deze fitness-waarden liggen tussen -30 en 30. Op die manier kunnen succesvolle maten de minder succesvolle moeilijk overwelmen (Biles, 1994).

Hoewel dit een goed systeem is om muziek te beoordelen, zorgt het voor een enorme vertraging aangezien de mentor steeds naar het fragment moet luisteren. Hierdoor komt er ook een limiet op de grootte van de populaties, het aantal generaties enzoverder.

3.3.5 Genetische operatoren

Een genetisch algoritme heeft operatoren nodig om de populatie te laten evolueren. In deze korte sectie bespreken we hoe de operatoren van GenJam werken.

De *initiële populatie* wordt willekeurig gegenereerd, aan de hand van bepaalde kansen. Bijvoorbeeld: de kans dat een noot blijft duren is $\frac{5}{24}$ (Biles, 1994).

De *selectie en vervanging* gebeuren door een aangepaste vorm van toernooiselectie (zie sectie 2.4.3). Er worden willekeurig vier individuen gekozen, deze vormen samen een *familie*. De twee beste individuen van een familie vormen de *ouders* en de twee slechtste worden vervangen door de kinderen die voortkomen uit de combinatie van de ouders. Deze *combinatie* gebeurt door een standaard one-point-crossover (zie sectie 2.4.3). Een van de kinderen wordt behouden een het andere kind wordt *gemuteerd*. Enkele mogelijke mutaties op zin- en op maatniveau zijn voorgesteld in tabel 3.1 (Biles, 1994).

Mutatie-operator	Gemuteerde maat
Geen	9 7 0 5 7 15 15 0
Omkering	0 15 15 7 5 0 7 9
Rotatie rechts	15 15 0 9 7 5 7
Inverteren (15-waarde)	6 8 15 10 8 0 0 15
Transponeren (vb +3)	12 10 0 8 10 15 15 0

Mutatie-operator	Gemuteerde zin
Geen	57 57 11 38
Omkering	38 11 57 57
Rotatie rechts (vb.3)	57 11 38 57

Tabel 3.1: Enkele mutatie-operatoren van GenJam(Biles, 2002)

3.3.6 Resultaten

GenJam geeft samen met John A. Biles optredens en lijkt vrij goede muziek te produceren. Biles is zich bewust van de fitness bottleneck en heeft een poging gedaan om deze te elimineren, door de behoefte aan fitness te elimineren. Dit werd op de volgende manier gedaan (Biles, 2001):

- de initiële populatie werd 'gevoed' met melodische ideeën uit een database van bestaande jazz-fragmenten.
- er werd gebruik gemaakt van een intelligent cross-over operator.

Het resultaat hiervan is '*Autonomous GenJam*' (Biles, 2001). De resultaten van dit nieuwe programma werden gekarakteriseerd als interessanter, meer coherent, menselijker en minder repetitief. Hoewel Autonomous GenJam betere muziek genereert is het echter strikt genomen geen genetisch algoritme meer (Biles, 2001), aangezien het geen gebruik maakt van fitness.

3.4 SIEDP

3.4.1 Inleiding

Het 'Intelligent System for Piano Learning Aid', ofwel SIEDP, is een ander soort CAC-toepassing. SIEDP componeert geen muziek, maar kan worden gebruikt als hulpmiddel bij het componeren. Het systeem berekent namelijk de beste vingerzetting om een muziekstuk op piano te spelen. Op die manier kan een componist snel zien of een geschreven stuk kan gespeeld worden op een piano.

De primaire doelstelling van SIEDP is echter niet om een hulpmiddel te zijn bij het componeren, maar om beginnende pianisten te helpen. Beginnende muzikanten verliezen vaak snel hun motivatie omdat ze geen rekening houden met de correcte vingerzettingen. Muziekstukken worden zo moeilijker speelbaar en dit is vaak frustrerend (Viana en de Morais Júnior, 2003).

3.4.2 Werking van SIEDP

SIEDP is ontstaan uit de samenwerking tussen professionele muzikanten, elektrische ingenieurs en informatici. Het bestaat uit twee hoofdcomponenten (Viana en de Morais Júnior, 2003):

- applicatiesoftware die de optimale vingerzetting bepaald.
- een robotisch hand die de melodie speelt, volgens de gekozen vingerzetting.

Wanneer we de beste vingerzetting van een melodie willen laten berekenen, volstaat het om de melodie in MIDI-vorm door te geven aan SIEDP. Het programma gaat vervolgens door middel van een genetisch algoritme (zie sectie 2.4.3) de beste oplossing proberen vinden. De beste oplossing is de vingerzetting waarbij een minimale spierinspanning nodig is. Dit is dus diegene met de kleinste vingerafstand (Viana en de Morais Júnior, 2003).

SIEDP werkt met populaties die bestaan uit 50 vingerzettingen. Aangezien het hier gaat om een genetisch algoritme, is het ook hier weer nodig om een fitness-functie te definiëren. Deze functie is een maat voor de 'afstand' van de vingerzetting van een gegeven melodie. Op basis van deze waarde gaat het algoritme goede individuen selecteren uit de populatie. Door selectie, crossover en mutatie gaat de populatie evolueren en zo hopelijk tot een goede oplossing komen.

3.4.3 Resultaten

Als output geeft SIEDP een opsomming met de tien beste vingerzettingen. De eindgebruiker kan dan zelf nog kiezen welke hij wil gebruiken. In de vorige versie van SIEDP was er een probleem met de convergentie. Het duurde lang voordat men een goede oplossing vond. Dit probleem ontstond doordat er veel 'klonen' in de populatie zaten, of met andere woorden, bepaalde oplossingen kwamen meer dan één keer voor in de populatie. In de laatste nieuwe versie is dit opgelost door een routine in te voeren die klonen verwijdert. Hierdoor komt men sneller tot een goede oplossing (Viana en de Morais Júnior, 2003).

DEEL II

Toepassing: Tabu Search voor de optimalisatie van muzikale fragmenten

4 Een tabu search algoritme voor het optimaliseren van muzikale fragmenten

4.1 Inleiding

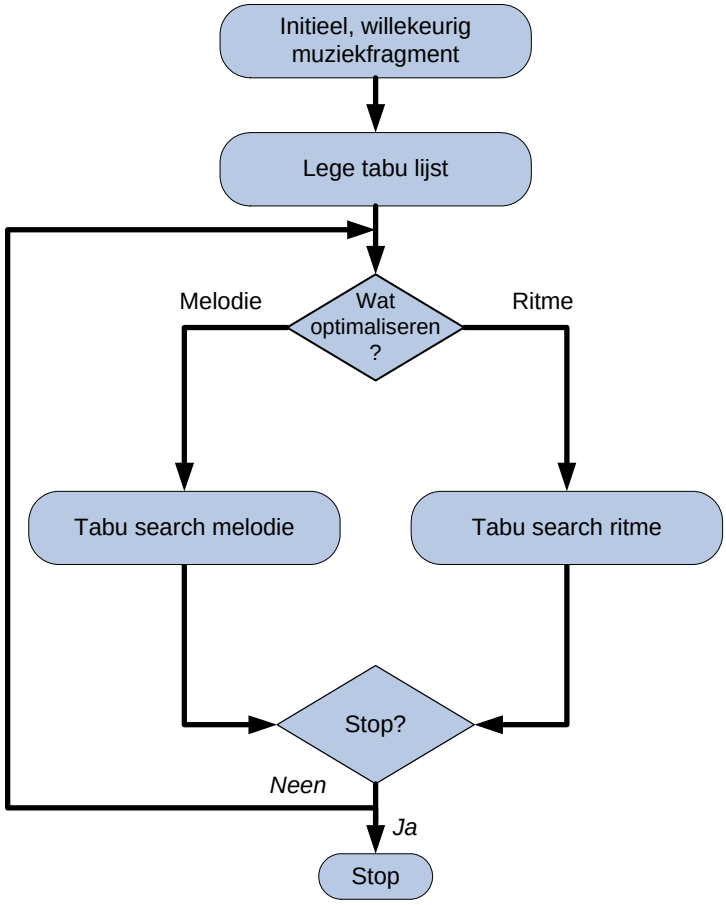
> *. . . les calculs n'ont cessé d'accompagner la musique tout au long de son histoire, et dans toutes les civilisations . . .*
>
> (Boulez, 1985), zoals aangehaald in (Harris et al., 1993)

We hebben zelf een eenvoudige CAC-toepassing geschreven die in staat is muziekfragmenten te optimaliseren. In dit deel bespreken we het zelf ontworpen Pascalprogramma *'TabuMusic'*, waarvan de broncode beschikbaar is in bijlage D en op de cd-rom. Op de bijhorende cd-rom staat tevens een gecompileerde versie voor win32 en linux. Ons hoofddoel bij het maken van TabuMusic was een algoritme te ontwerpen dat een willekeurig gegenereerd muziekstuk optimaliseert met behulp van een tabu search-algoritme. Hierbij wordt er een score toegekend aan de muziek. Vervolgens wordt de enorme zoekruimte van mogelijke noten en ritmes op een gerichte manier verkend, om zo hopelijk tot een goede oplossing te komen.

In de bestaande literatuur omtrent Computer Assisted Composing bestaan er weinig of geen toepassingen die gebruik maken van tabu search. Het leek ons daarom interessant om na te gaan of een tabu search algoritme ook kan worden gebruikt om een muziekstuk te componeren. Zoals in hoofdstuk 3 bleek, worden andere metaheuristieken, zoals bijvoorbeeld genetische algoritmen, reeds veel gebruikt in dit vakgebied. Onze bedoeling is om na te gaan of een tabu search algoritme in staat is om snel een goed muziekstuk te vinden, gegeven enkele beperkingen.

We gaan homofone muziek optimaliseren, dit wil zeggen dat in het te optimaliseren muziekstuk steeds maar één noot tegelijk klinkt. Deze vereenvoudiging zorgt onder meer voor een gemakkelijkere representatie en een simpelere doelfunctie, zoals duidelijk wordt secties 4.2 en 4.4.

De structuur van de ontwikkelde toepassing is voorgesteld in figuur 4.1. We vertrekken van een initieel willekeurig muziekfragment en een lege tabulijst. Vervolgens kunnen we steeds kiezen of we het ritme of de melodie willen optimaliseren. De gebruiker specificeert op voorhand hoeveel keer hij het ritme en de melodie wil optimaliseren en in welke volgorde. In wat volgt worden de verschillende stappen meer in detail uitgelegd.



Figuur 4.1: Schematische voorstelling van de ontwikkelde toepassing

4.2 Voorstellen van muziek

Een eerste stap is het weergeven van muziek. Hoewel de *CHARM*-notatie (zie sectie 3.2.1) een goede algemene voorstellingswijze vormt hebben we er niet voor gekozen om ze te gebruiken. Voor onze toepassing is het belangrijk om muziek op een simpele manier voor te stellen. Een eenvoudige structuur is hiervoor beter geschikt, zodat we een duidelijk en gestructureerd beeld krijgen van wat er op elk tijdstip gebeurd in het muziekstuk. Aangezien we geen gebruik maken van ingewikkelde regels in verband met grammatica, zou de *CHARM*-methode onnodige complexiteit aan het model toevoegen.

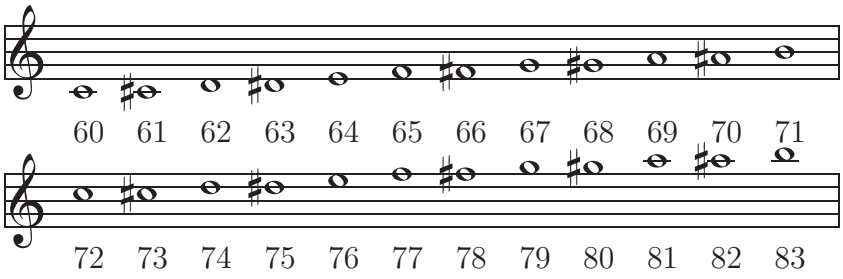
We hebben zelf een simpele matrixvoorstelling ontworpen om muziek weer te geven. Een muziekstuk wordt hierbij voorgesteld als een eenvoudige $(2 \times m)$-matrix. Waarbij m de lengte van het fragment is, uitgedrukt in 16de noten. De noot in kolom j hoort bij het j-de tijdstip.

In de **eerste rij** wordt elke noot voorgesteld door een integer waarde. Deze waarde is de *midi-waarde* van de noot. Een bepaalde toonhoogte wordt voorgesteld door een integer, zo krijgt de lage do de waarde 60 toegewezen. Een verhoging van een halve toon staat gelijk aan de vorige midi-waarde + 1 (Miranda, 2001). De oorspronkelijke bedoeling van deze voorstellingswijze was dat het voor een gemakkelijkere conversie naar het MIDI-formaat zou zorgen. Achteraf is deze conversie echter gebeurd via de ABC-notatie (zie sectie 4.7.3).

De **tweede rij** bevat steeds een 1 of 0, al naar gelang de noot wel of niet blijft voortduren op het volgende tijdstip. Een voorbeeld van de ontwikkelde matrixvoorstelling wordt gegeven in figuur 4.4.

Om realistisch te blijven moet er ook een beperking komen met betrekking tot het aantal *mogelijke toonhoogtes*. Het toonbereik van het muziekstuk gaat van de lage do tot en met de hoge si, en omvat twee volledige octaven. Zo kunnen we ongeveer evenveel klanken weergeven als een klassieke fluit (hoewel deze net nog iets hoger kan gaan). De volledige range samen met de overeenkomstige midi-waarden zijn voorgesteld in figuur 4.2.

Een volgende beperking die nodig is, heeft betrekking op het ritme. Tijdens een muziekstuk houdt men het ritme aan door te 'tellen'. De duur van de noten is gebaseerd op deze tellen en op de maatstaf. Wij hebben als maatstaf $\frac{4}{4}$ genomen, dit wil zeggen dat de totale lengte van een maat bestaat uit 4 noten met duur

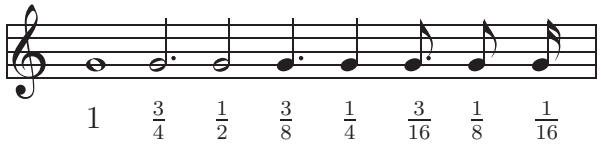


Figuur 4.2: Gebruikte noten en hun MIDI-waarde

$\frac{1}{4}$ (nootduur $\frac{1}{4}$ komt dus overeen met 1 tel). Deze noten kunnen natuurlijk nog verder worden opgesplitst in vb. 2 achtste-noten ($\frac{1}{8} + \frac{1}{8} = \frac{1}{4}$). De duur van 1 tel (1 vierdenoot) is relatief en afhankelijk van het gekozen tempo (dit kan vb. zijn $\frac{1}{4} = 120$ keer per minuut). Er bestaan verschillende soorten noten die elk een andere duur hebben:

- Een achtste-noot $\frac{1}{8}$
- Een kwartnoot $\frac{1}{4}$
- Een halve noot $\frac{1}{2}$
- ...

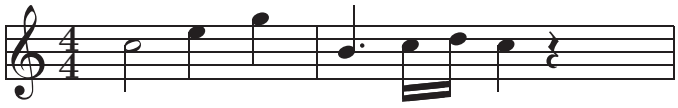
De basis-nootduren zijn voorgesteld in figuur 4.3. Een basisnootduur kan aangepast worden door er een puntje naast te zetten, dan doet men zijn duur×1.5. Er zijn nog andere aanpassingen mogelijk om nog meer verschillende nootduren te kunnen voorstellen.



Figuur 4.3: Ritmedeeltjes

Om de voorstelling van het te optimaliseren stuk te vereenvoudigen nemen we als kleinst mogelijke ritmedeeltje de *16e noot* en als langste de hele noot. Samenvattend kunnen we stellen dat een muziekstuk van n maten zal bestaan uit maximaal $n \times 16$ noten. Het volledige muziekstuk komt in een $2 \times (n \times 16)$-matrix. Een voorbeeld hiervan is uitgewerkt in figuur 4.4, waar een kort fragment van Sonate KV 545 van Mozart in matrixvorm is beschreven.

In deze matrix hebben de elementen volgende betekenis:



$$A = \begin{bmatrix} 72 & 72 & 72 & 72 & 72 & 72 & 72 & 72 & 76 & 76 & 76 & 76 & 79 & 79 & 79 & 79 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 71 & 71 & 71 & 71 & 71 & 71 & 72 & 73 & 71 & 71 & 71 & 71 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Figuur 4.4: Wolfgang Amadeus Mozart - Sonate KV 545

- Element a_{1k} van de matrix geeft de midi-waarde van de noot op de k-de positie. Wanneer het om een rust gaat is deze waarde -1.
- Element a_{2k} van de matrix geeft aan of de k-de noot blijft voortduren (1) of niet (0). Een 0 duidt erop dat er ofwel een *andere* noot/rust volgt ofwel dat de noot *herhaald* wordt (en dus niet blijft voortduren). Elke kolom staat voor een $\frac{1}{16}$-noot. Dit is dus de kleinste ritmische eenheid die we kunnen voorstellen.

4.3 Initiële willekeurige muziek

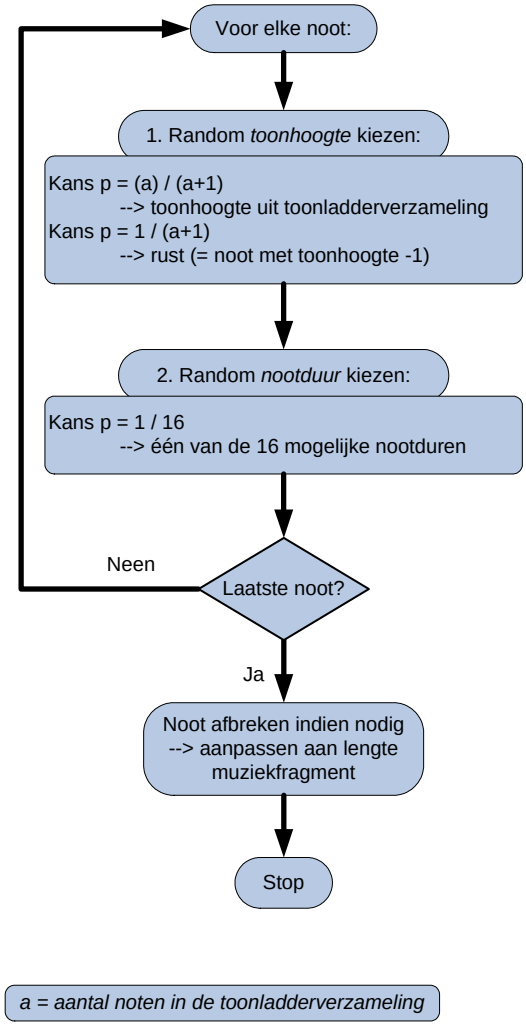
We vertrekken van een willekeurig muziekfragment, dat we later gaan optimaliseren. Hoe genereren we deze muziek? Dit zal gebeuren op basis van een aantal parameters die de gebruiker doorgeeft aan het systeem:

- de toonaard.
- het aantal maten muziek dat hij wil genereren.

Wat betreft de *toonaard* kunnen we opmerken dat alle mogelijke majeur-toonaarden mogelijk zijn. Een uitbreiding naar mineur of andere speciale toonaarden kan later gemakkelijk worden gerealiseerd. Daarvoor hoeven we enkel de manier waarop de '*toonladderverzameling*' wordt gegenereerd aan te passen. Deze toonladderverzameling is een simpele matrix die alle mogelijke noten bevat uit de toonladder. Deze matrix wordt helemaal in het begin van het programma gegenereerd.

De parameter '*aantal maten* muziek die de gebruiker wil genereren', spreekt voor zich. De gebruiker kiest op voorhand de lengte van het muziekfragment. Dit blijft constant gedurende het optimalisatieproces.

Het genereren van een *willekeurig muziekstuk* gebeurt zoals is voorgesteld in figuur 4.5. Voor elke noot wordt er eerst een *toonhoogte* gekozen. Een rust wordt beschouwd als een noot met toonhoogte -1, deze is niet opgenomen in de toonladderverzameling. De kans p om een bepaalde toonhoogte te kiezen is $p = 1/(\text{aantal noten in toonladderverzameling} + 1)$. Vervolgens krijgt elke noot een *duur*. We hebben de keuze uit 16 mogelijke nootduren (zie sectie 4.2). De kans q op een bepaalde nootduur is dus $q = \frac{1}{16}$.



Figuur 4.5: Algoritme generatie willekeurig muziekfragment

Nadat het willekeurig muziekstuk is gegenereerd zal er een score moeten toegekend worden aan het fragment. Deze geeft de muzikale kwaliteit weer van de gegenereerde muziek.

4.4 Muzikale kwaliteit van een fragment

We hebben gekozen voor een geautomatiseerd doelfunctie, omdat deze de meeste voordelen biedt. Een doelfunctie *berekenen* is zeer snel en geeft een objectief resultaat.

Als criteria voor het meten van de kwaliteit van homofone muziek vinden we bij verschillende auteurs (Snarrenberg, 2004; Schell, 2002; Papadopoulos en Wiggins, 1998) maatstaven terug die sterk met elkaar verband houden o.a. geconnecteerdheid, variëteit, intervalgrootte, consonantie, toonbereik,... Wij gebruiken de criteria van Towsey omdat deze de meeste van de vorige omvatten (Towsey et al., 2001).

De gebruikte criteria kunnen elk worden voorgesteld in breukvorm: (Towsey et al., 2001):

Pitch variety De verhouding tussen het aantal verschillende toonhoogtes en het aantal noten. Dit criterium geeft de diversiteit in het aantal klanken weer.

$$\text{criterium } 1 = \frac{\text{aantal verschillende toonhoogtes}}{\text{aantal noten}}$$

Dissonant intervals De fractie dissonante intervallen. De dissonantiewaarde van de verschillende intervallen (gemeten in halve tonen en de rusten genegeerd), worden gegeven in tabel 4.1.

Interval	Dissonantiewaarde
0, 1, 2, 3, 4, 5, 7, 8, 9, 12	0.0
10	0.5
6, 11, ≥ 13	1.0

Tabel 4.1: Dissonantiewaarden

$$\text{criterium } 2 = \frac{\sum \text{alle dissonantiewaarden}}{\text{aantal intervallen(=aantal noten}-1)}$$

Contour direction De verhouding tussen het aantal stijgende intervallen en het totaal aantal intervallen. Rusten worden genegeerd.

$$\text{criterium } 3 = \frac{\sum \text{alle stijgende intervallen}}{\sum \text{alle intervallen}}$$

Criterium	Gemiddelde	Standaardafwijking
Pitch variety	0.27	0.11
Dissonant intervals	0.01	0.02
Contour direction	0.49	0.06
Contour stability	0.40	0.11
Rhythmic variety	0.24	0.07
Rhythmic range	0.32	0.11

Tabel 4.2: De belangrijkste criteria samen met hun gemiddelde en standaardafwijking

Contour stability De proportie intervallen die in dezelfde richting evolueren als het vorige interval. Rusten worden genegeerd.

$$\text{criterium } 4 = \frac{\text{aantal opeenvolgende intervallen die in dezelfde richting evolueren}}{\text{aantal intervallen}-1}$$

Rhythmic variety De ritmische coherentie: de graad van gebruik van de 16 mogelijke ritmes.

$$\text{criterium } 5 = \frac{\text{aantal verschillende nootduren}}{16}$$

Rhythmic range Het ritmische bereik

$$\text{criterium } 6 = \frac{\text{max nootduur}-\text{min nootduur}}{16}$$

Deze criteria alleen zijn echter niet voldoende. We moeten ook hun optimale waarde weten. Towsey heeft hiernaar een onderzoek gedaan. Hij heeft een aantal bekende muziekstukken uit verschillende muziekstijlen ingevoerd in de computer en de waarde berekend voor elk bovenstaand criterium en een aantal andere criteria (Towsey et al., 2001). De belangrijkste hiervan (die met de kleinste standaardafwijking) zijn diegene die net besproken zijn geweest. Ze staan opgesomd in tabel 4.2, samen met hun gemiddelde waarde en standaardafwijking.

We vormen de criteria om zodat ze minimaal zijn wanneer hun optimale waarde (het gemiddelde in tabel 4.2) bereikt wordt. Om de doelfunctie concreet te berekenen nemen we een gewogen gemiddelde. De gebruiker kan de wegingscoëfficënten als parameter doorgeven aan het programma. In sectie 5.2 bespreken we de keuze van deze wegingscoëfficënten. We krijgen dan volgende (gestandaardiseerde) functie:

$$\text{score} = a_1 \frac{|\text{criterium } 1 - 0.27|}{0.11} + a_2 \frac{|\text{criterium } 2 - 0.01|}{0.02} + a_3 \frac{|\text{criterium } 3 - 0.49|}{0.06} + a_4 \frac{|\text{criterium } 4 - 0.40|}{0.11} + a_5 \frac{|\text{criterium } 5 - 0.24|}{0.07} + a_6 \frac{|\text{criterium } 6 - 0.32|}{0.11}$$

met a_i = wegingscoëfficiënt van criterium i

Deze functie is optimaal wanneer ze nul is. Daarom is het doel van ons tabu search-algoritme om deze functie te *minimaliseren*.

4.5 Local search operators

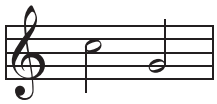
We optimaliseren de willekeurig gegenereerde muziek met behulp van een tabu search algoritme. Zoals in sectie 2.4.4 is uiteengezet, hebben we steeds een local search-operator nodig om onze neighbourhood te kunnen berekenen. Enkele voorbeelden van mogelijke moves kwamen reeds aan bod in sectie 3.3. In wat volgt bespreken we de local search-operators van de melodie en het ritme.

4.5.1 Optimalisatie melodie

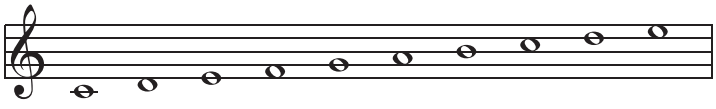
Als *move* kozen we het vervangen van een noot door een noot met dezelfde lengte, maar met een andere toonhoogte. De *neighbourhood* van een muziekfragment bestaat uit alle fragmenten waarbij telkens één noot wordt vervangen. Deze noot wordt in de verschillende fragmenten vervangen door alle mogelijke noten uit de gekozen toonladderverzameling. We proberen alle noten te vervangen.

Een voorbeeld maakt dit iets concreter.

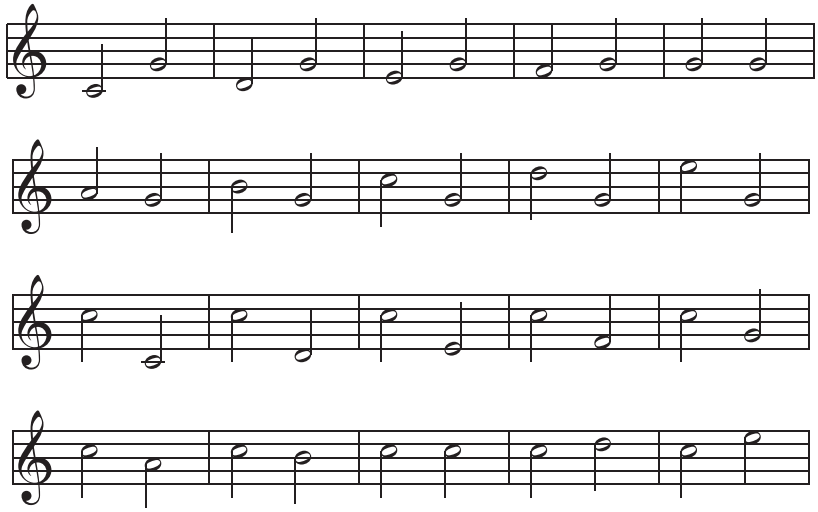
Stel je wil volgend (zeer) kort fragment optimaliseren:



Het is een fragment in C, wat wil zeggen dat er geen ♯ of ♭ voorkomen in de toonladderverzameling. De toonladderverzameling is hieronder voorgesteld. We merken op dat ze alle noten bevat die mogen voorkomen in het fragment en die tot de toonladder behoren.



Hieronder wordt de hele neighbourhood voorgesteld. Een move wordt weergegeven als één maat in de figuur.



4.5.2 Optimalisatie ritme

Een *move* is het veranderen van de duur van een noot door eender welke andere mogelijke duur. De *neighbourhood* van een muziekfragment bestaat dus uit alle fragmenten waarbij telkens de lengte van één noot wordt aangepast. We proberen alle noten aan te passen.

Een voorbeeld maakt dit iets concreter.

We optimaliseren een zeer kort fragment dat bestaat uit drie noten, zie tabel 4.3. Om te verduidelijken hoe we een move maken stellen we het fragment in tabelvorm voor:

1ste noot	duur	2de noot	duur	3de noot	duur
sol	$\frac{8}{16}$	la	$\frac{2}{16}$	do	$\frac{6}{16}$

Tabel 4.3: Voorbeeldfragment

De nootduur wordt steeds voorgesteld als een veelvoud van $\frac{1}{16}$. Dit komt omdat we $\frac{1}{16}$ als kleinste ritmedeeltje hebben beschouwd (zie sectie 4.2). De som van alle nootduren in één maat moet steeds $\frac{16}{16}$ zijn. Aangezien ons voorbeeldfragment maar uit één maat bestaat moet de som van alle nootduren steeds $\frac{16}{16}$ zijn. Op deze manier blijft de lengte van de muziek steeds even lang.

Een move is het aanpassen de nootduur van één noot. Voor elke noot zullen we alle mogelijke duren proberen. De *neighbourbood* die hierbij hoort is voorgesteld in tabel 4.4.

We zien echter dat de som van alle nootduren groter is dan $\frac{16}{16}$. Er zijn twee mogelijke gevallen:

1. lengte $> \frac{16}{16} \rightarrow$ kap de laatste noot af.
2. lengte $< \frac{16}{16} \rightarrow$ vul het muziekstuk achteraan aan met de verkorte noot.

Dit is uitgewerkt in tabel 4.5.

4.6 Tabu-structuren

De algoritmen voor het optimaliseren van het ritme en de melodie lijken sterk op elkaar. In figuur 4.6 worden ze schematisch weergegeven. In beide gevallen gaat het om een tabu search algoritme.

4.6.1 Optimalisatie melodie

De optimalisatie van de melodie gebeurt door het oproepen van de functie *noot-transponeren*. De werking van deze tabu search functie is schematisch voorgesteld in figuur 4.6.

De allereerste keer vertrekken we van het willekeurig gegenereerde muziekstuk en een lege tabulijst. De gebruiker kan in het begin volgende parameters in verband met de melodie doorgeven aan het systeem:

- de lengte van de tabulijst, ofwel de tabu tenure.
- het aantal iteraties, hoeveel keer willen we het algoritme doorlopen en een move maken?

Eerst wordt de neighbourhood van het originele fragment s gegenereerd zoals is uiteengezet in de vorige sectie. Vervolgens wordt de muzikale kwaliteit van alle oplossingen uit de neighbourhood berekend. Hoe goed is hun score? Wanneer een oplossing beter of even goed is dan de score van s, dan wordt dit onze nieuwe oplossing t. Dit zal echter enkel gebeuren als de move niet op de *tabulijst* staat. In de tabulijst houden we bij welke noten onlangs vervangen zijn en welke midi-waarde

duur n1	duur n2	duur n3	duur n1	duur n2	duur n3
1/16	2/16	6/16	8/16	9/16	6/16
2/16	2/16	6/16	8/16	10/16	6/16
3/16	2/16	6/16	8/16	11/16	6/16
4/16	2/16	6/16	8/16	12/16	6/16
5/16	2/16	6/16	8/16	13/16	6/16
6/16	2/16	6/16	8/16	14/16	6/16
7/16	2/16	6/16	8/16	15/16	6/16
8/16	2/16	6/16	8/16	16/16	6/16
9/16	2/16	6/16	8/16	2/16	1/16
10/16	2/16	6/16	8/16	2/16	2/16
11/16	2/16	6/16	8/16	2/16	3/16
12/16	2/16	6/16	8/16	2/16	4/16
13/16	2/16	6/16	8/16	2/16	5/16
14/16	2/16	6/16	8/16	2/16	6/16
15/16	2/16	6/16	8/16	2/16	7/16
16/16	2/16	6/16	8/16	2/16	8/16
8/16	1/16	6/16	8/16	2/16	9/16
8/16	2/16	6/16	8/16	2/16	10/16
8/16	3/16	6/16	8/16	2/16	11/16
8/16	4/16	6/16	8/16	2/16	12/16
8/16	5/16	6/16	8/16	2/16	13/16
8/16	6/16	6/16	8/16	2/16	14/16
8/16	7/16	6/16	8/16	2/16	15/16
8/16	8/16	6/16	8/16	2/16	16/16

Tabel 4.4: Voorbeeld neighbourhood

duur n1	duur n2	duur n3	nieuwe n4	duur n1	duur n2	duur n3	nieuwe n4
$\frac{1}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	$\frac{7}{16}$ noot 1	$\frac{8}{16}$	$\frac{8}{16}$	/	
$\frac{2}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	$\frac{6}{16}$ noot 1	$\frac{8}{16}$	$\frac{8}{16}$	/	
$\frac{3}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	$\frac{5}{16}$ noot 1	$\frac{8}{16}$	$\frac{8}{16}$	/	
$\frac{4}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	$\frac{4}{16}$ noot 1	$\frac{8}{16}$	$\frac{8}{16}$	/	
$\frac{5}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	$\frac{3}{16}$ noot 1	$\frac{8}{16}$	$\frac{8}{16}$	/	
$\frac{6}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	$\frac{2}{16}$ noot 1	$\frac{8}{16}$	$\frac{8}{16}$	/	
$\frac{7}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	$\frac{1}{16}$ noot 1	$\frac{8}{16}$	$\frac{8}{16}$	/	
$\frac{8}{16}$	$\frac{2}{16}$	$\frac{6}{16}$		$\frac{8}{16}$	$\frac{8}{16}$	/	
$\frac{9}{16}$	$\frac{2}{16}$	$\frac{5}{16}$		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{1}{16}$	$\frac{5}{16}$ noot 3
$\frac{10}{16}$	$\frac{2}{16}$	$\frac{4}{16}$		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{2}{16}$	$\frac{4}{16}$ noot 3
$\frac{11}{16}$	$\frac{2}{16}$	$\frac{3}{16}$		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{3}{16}$	$\frac{3}{16}$ noot 3
$\frac{12}{16}$	$\frac{2}{16}$	$\frac{2}{16}$		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{4}{16}$	$\frac{2}{16}$ noot 3
$\frac{13}{16}$	$\frac{2}{16}$	$\frac{1}{16}$		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{5}{16}$	$\frac{1}{16}$ noot 3
$\frac{14}{16}$	$\frac{2}{16}$	/		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	
$\frac{15}{16}$	$\frac{1}{16}$	/		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	
$\frac{16}{16}$	/	/		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	
$\frac{8}{16}$	$\frac{1}{16}$	$\frac{6}{16}$	$\frac{1}{16}$ noot 2	$\frac{8}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	
$\frac{8}{16}$	$\frac{2}{16}$	$\frac{6}{16}$		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	
$\frac{8}{16}$	$\frac{3}{16}$	$\frac{5}{16}$		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	
$\frac{8}{16}$	$\frac{4}{16}$	$\frac{4}{16}$		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	
$\frac{8}{16}$	$\frac{5}{16}$	$\frac{3}{16}$		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	
$\frac{8}{16}$	$\frac{6}{16}$	$\frac{2}{16}$		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	
$\frac{8}{16}$	$\frac{7}{16}$	$\frac{1}{16}$		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	
$\frac{8}{16}$	$\frac{8}{16}$	/		$\frac{8}{16}$	$\frac{2}{16}$	$\frac{6}{16}$	

Tabel 4.5: Voorbeeld aangepaste neighbourhood

ze hadden. Stel dat we op een bepaald ogenblik de volgende tabulijst hebben met als tenure 5:

$$A = \begin{bmatrix} 173 & 5 & 0 & 0 & 0 \\ 65 & 60 & 0 & 0 & 0 \end{bmatrix}$$

In het voorbeeld werden reeds twee noten vervangen. De eerste noot die vervangen werd begon op de vijfde tijd (vijfde positie in de ontworpen matrixvoorstelling). Deze noot had als midi-waarde 60, het was dus een do. De volgende vervangen noot had als midi-waarde 65 en begon op de 173ste tijd.

In TabuMusic is ook een *aspiratieconditie* ingebouwd. We houden steeds de allerbeste oplossing bij die we gevonden hebben. Wanneer we een oplossing u tegenkomen die beter is dan de allerbeste, dan nemen we deze steeds aan. Zelfs als de move op de tabulijst staat wordt hij toch gedaan omdat we onze allerbeste oplossing kunnen verbeteren.

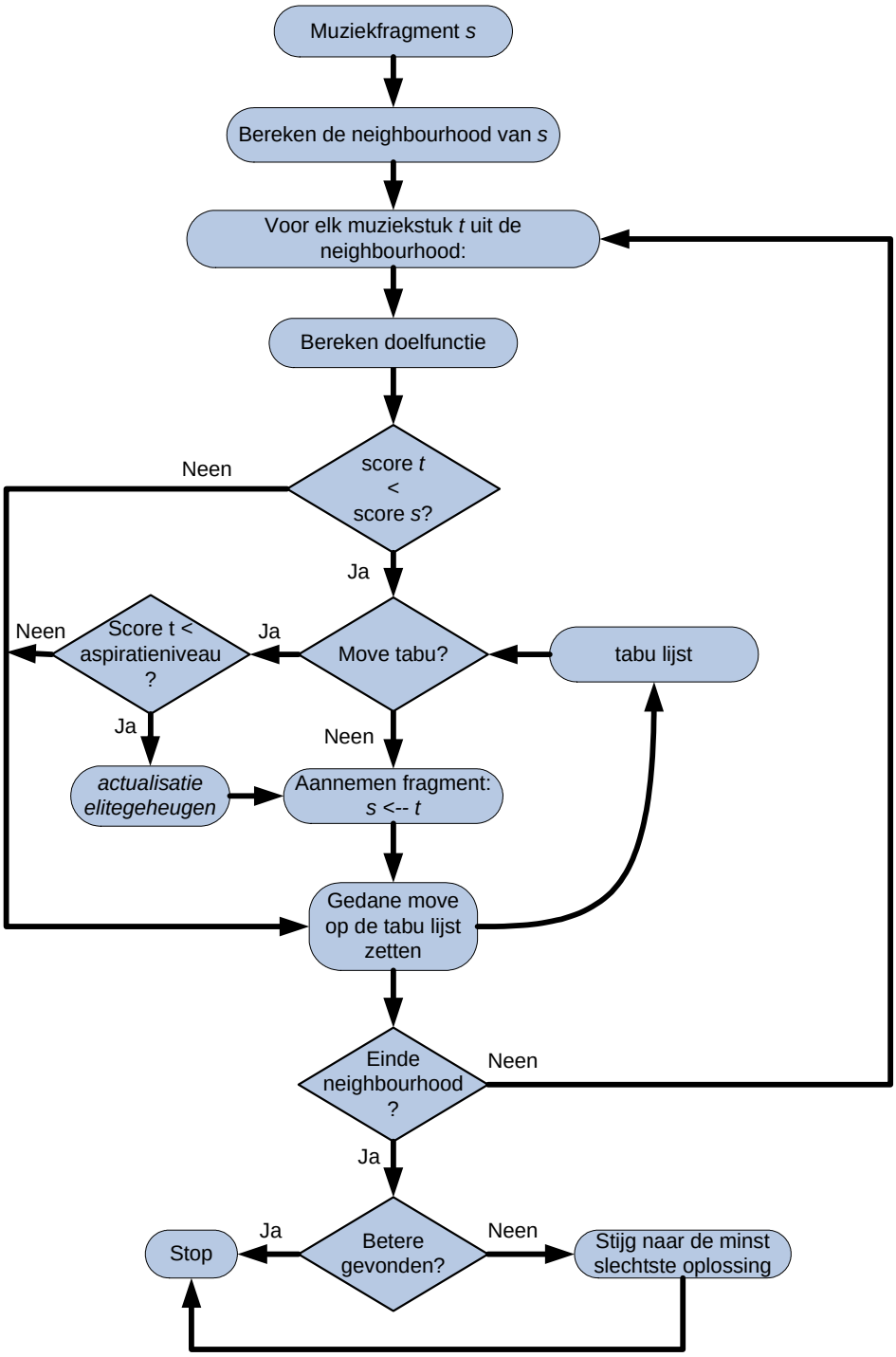
Wanneer er een oplossing t of u wordt aangenomen, dan komt de gedane move op de eerste plaats van de tabulijst te staan en schuift alles een plaats op. In het geval dat er geen betere oplossing gevonden is, stijgen we naar de minst slechte oplossing.

Om de volgende move te maken vertrekken we van de laatst aangenomen oplossing. Hiervan berekenen we weer de neighbourhood, enzoverder. De gebruiker kan in het begin specifiëren hoeveel keer de procedure herhaald wordt.

4.6.2 Optimalisatie ritme

De optimalisatie van het ritme gebeurt door het oproepen van de functie *ritmeverbeteren*. De werking van deze tabu search functie is schematisch voorgesteld in figuur 4.6. De gebruiker geeft weer dezelfde parameters door als bij de optimalisatie van de melodie.

Net zoals bij de melodie vertrekken we van het willekeurig gegenereerde muziekstuk en een lege tabulijst. Dan wordt de neighbourhood van het originele fragment s gegenereerd zoals is uiteengezet in sectie 4.5. Vervolgens wordt de muzikale kwaliteit berekend van alle oplossingen uit de neighbourhood. Hoe goed is hun score? Wanneer een oplossing beter of even goed is dan de score van s, dan wordt dit onze nieuwe oplossing t. Dit zal, net zoals bij de melodie, enkel gebeuren als de move niet op de *tabulijst* staat. Wanneer een oplossing beter of even goed is dan de vorige,



Figuur 4.6: Optimalisatie van de melodie en het ritme

dan zal ze worden aangenomen als de move niet op de *tabulijst* staat. In de tabulijst houden we bij welke noten onlangs vervangen zijn. Stel dat we op een bepaald ogenblik de volgende tabulijst hebben met als tenure 5:

$$A = \begin{bmatrix} 10 & 8 & 2 & 0 & 0 \end{bmatrix}$$

Er werden reeds drie noten vervangen. De noot die eerst vervangen werd was de tweede noot. Let op, dit is niet de noot die op de tweede tijd begint, zoals bij de melodie. Dit is de tweede noot van het muziekstuk. Vervolgens werd ook de achtste en de tiende noot vervangen.

Ook voor de optimalisatie van het ritme is er een *aspiratieconditie* ingebouwd. We houden steeds de allerbeste oplossing bij die we gevonden hebben. Wanneer we een oplossing u tegenkomen die beter is dan de allerbeste, dan nemen we deze steeds aan. Zelfs als de move op de tabulijst staat wordt hij toch gedaan omdat we onze allerbeste oplossing kunnen verbeteren.

Wanneer er een oplossing t of u wordt aangenomen, dan komt de gedane move op de eerste plaats van de tabulijst te staan en schuift alles een plaats op. In het geval dat er geen betere oplossing gevonden is, stijgen we naar de minst slechte oplossing.

Om de volgende move te maken vertrekken we van de laatst aangenomen oplossing. Hiervan berekenen we weer de neighbourhood, enzoverder. De gebruiker kan in het begin specifiëren hoeveel keer de procedure herhaald wordt.

4.7 Implementatiedetails

In deze sectie wordt dieper ingegaan op de specifieke implementatiedetails van TabuMusic. Welke parameters kunnen we doorgeven, in welke taal is het geschreven en hoe kunnen we de gecomponeerde muziek beluisteren?

4.7.1 Parameters

Wanneer we het programma 'TabuMusic' oproepen zal het eerst het bestand 'inputmuziek.txt' aanroepen. Hierin staan volgende parameters gespecificeerd:

Integer	Toonladder
60	C
61	C♯
62	D
63	D♯
64	E
65	F
66	F♯
67	G
68	G♯
69	A
70	A♯
71	B

Tabel 4.6: De integer-waarden van de verschillende toonladders

- De toonladder van de te genereren muziek. Dit wordt aangegeven door een integer. Tabel 4.6 geeft een overzicht van de mogelijke waarden. Het gaat hier steeds om de majeur toonladder.
- Het aantal maten muziek dat de gebruiker wil genereren.
- De bestandsnaam waarin de gebruiker de willekeurige muziek wil opslaan. Dit is bij voorkeur een .abc-bestand.
- De bestandsnaam waarin de gebruiker de geoptimaliseerde muziek wil opslaan. Dit is bij voorkeur een .abc-bestand.
- De bestandsnaam waarin de tekstoutput van TabuMusic komt.
- Het aantal keer dat de gebruiker wil itereren per optimalisatie. Dit is het aantal keer dat er een neighbourhood zal worden gegenereerd en onderzocht per tabu search.
- De lengte van de tabulijst van de melodie, maximaal 1000.
- De lengte van de tabulijst van het ritme, maximaal 1000.
- De zes wegingscoëfficiënten. één voor elk criterium van de doelfunctie.
- De sequentie van de optimalisaties: dit is een opeenvolging van ofwel de melodie (1) of het ritme (2) en kan er bijvoorbeeld als volgt uitzien: 1 2 2 2 1 1

2 1 2.

- Het bestand wordt afgesloten met een 0.

TabuMusic leest al deze parameters in wanneer het wordt aangeroepen. Een voorbeeld van een inputmuziek-file is bijgevoegd in bijlage C.

4.7.2 Programmeertaal

In sectie 3.2.3 zijn enkele talen genoemd die speciaal ontworpen zijn voor CAC-toepassingen. Deze talen zijn echter zeer ingewikkeld, daarom hebben wij er geen gebruik van gemaakt. Wij kozen ervoor om de gestructureerde taal Pascal te gebruiken omdat zij volgende voordelen bood:

- het is een zeer snelle taal
- ze is zeer gestructureerd
- de syntax is relatief eenvoudig

Deze eigenschappen zorgen ervoor dat we de gestructureerde en snelle aard van onze oplossingsmethode (tabu search) niet verliezen in een warboel aan programeerobjecten. Bij Pascal zijn er echter geen ingebouwde functies om muziek om te zetten in geluid. In de volgende sectie wordt verklaard hoe we dit hebben opgelost.

4.7.3 De muziek beluisteren

Wanneer TabuMusic twee muziekstukken gecomponeerd heeft, een willekeurig en een geoptimaliseerd, is het interessant om ze aan een menselijk oor te onderwerpen. We willen weten of er een verbetering te horen is in het muziekstuk.

Er zijn verschillende bestandsformaten voor muziek: mp3, wav, midi,... Voor ons brengt het *midi* formaat de meeste voordelen mee. In sectie 1.4.3 werd de MIDI-standaard reeds meer in detail uitgediept. Voor ons is vooral van belang dat er zoiets bestaat als een MIDI-bestand. Dit is afspeelbaar op gelijk welke MIDI-speler.

Praktisch is niet gemakkelijk om een tekstbestand om te zetten naar een MIDI of .mid bestand. Er bestaan programma's die dit kunnen doen voor ons, zie (de Bruijn, 2004). We hebben echter gekozen om de muziek via een omweg om te zetten in midi, door gebruik te maken van de ABC-notatie. Deze notatie stelt elke noot

voor door een letter van A tot G (in kleine letters voor het 2e octaaf). Een rust wordt voorgesteld als een z. Deze letter wordt steeds gevolgd door een integer die de lengte aangeeft. Dit getal is steeds een veelvoud van de basisduur (bij ons $\frac{1}{16}$) (Mansfield, 2004). In figuur 4.7 wordt een muziekfragment in matrixvorm omgezet in de ABC-notatie.

$$A = \begin{bmatrix} 72 & 72 & 72 & 72 & 72 & 72 & 72 & 72 & 76 & 76 & 76 & 76 & 79 & 79 & 79 & 79 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 71 & 71 & 71 & 71 & 71 & 71 & 72 & 73 & 71 & 71 & 71 & 71 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$\Downarrow$

```
X:1
T:Gegenereerd muziekstuk
M:4/4
K:C
L:1/16
c8 e4 g4 B6 c1 d1 c4 z4
```

Figuur 4.7: Matrixvoorstelling omgezet in ABC-notatie

In de header van het ABC-bestand staat nog additionele informatie:

- X=1: duidt op het begin van het muziekstuk
- T: de titel van het muziekfragment
- M: de maatsoort
- K=C: de toonaard is C: er staat dus geen ♯ of ♭ vooraan
- L=1/16: de basisnootduur is $\frac{1}{16}$

De functie 'schrijfabc' van TabuMusic genereert als output een .abc-file. Er zijn allerlei toepassingen beschikbaar die een .abc-files kunnen omzetten naar een .mid-file en tegelijk bladmuziek genereren in .pdf. Een voorbeeld van een gratis beschikbaar programma met deze functionaliteit is *iABC*. Dit opensource-programma is gratis te downloaden via `http://abc.sourceforge.net/iabc/` (Walshaw, 2005), het staat tevens op de cd-rom.

Nu we de werking van TabuMusic besproken hebben, gaan we het in het volgende hoofdstuk verder testen.

5 Experimenten

I been searchin' . . .
Searchin' . . . oh yeah
Searchin' every which way

(Leiber en Stoller, 1957)

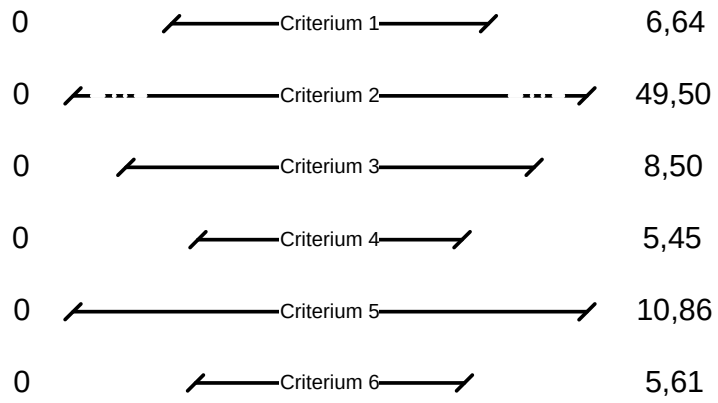
5.1 Inleiding

In dit laatste hoofdstuk gaan we TabuMusic evalueren. We staan eerst stil bij de wegingscoëfficiënten van de doelfunctie. Vervolgens vergelijken we resultaten van melodische en ritmische optimaliseringen. Tenslotte volgt een persoonlijke mening van de gegenereerde muziek. Alle testruns gebeurden met een Intel pentium IV met een kloksnelheid 1.4GHz en 512MB RAM-geheugen. Deze pc draait onder het operating system Windows XP.

5.2 Wegingscoëfficiënten

De doelfunctie is cruciaal bij het evalueren van muziek. Aangezien zij bestaat uit zes verschillende criteria (zie sectie 4.4), is het belangrijk om na te gaan hoe zwaar deze termen in de doelfunctie doorwegen. De doelfunctie is hieronder nog eens weergegeven:

$$\text{score} = a_1 \frac{|\text{criterium } 1 - 0.27|}{0.11} + a_2 \frac{|\text{criterium } 2 - 0.01|}{0.02} + a_3 \frac{|\text{criterium } 3 - 0.49|}{0.06} + a_4 \frac{|\text{criterium } 4 - 0.40|}{0.11} + a_5 \frac{|\text{criterium } 5 - 0.24|}{0.07} + a_6 \frac{|\text{criterium } 6 - 0.32|}{0.11}$$



Figuur 5.1: Minimale en maximale waarde van de 6 termen in de doelfunctie.

met a_i = wegingscoëfficiënt van criterium i

In figuur 5.1 is de minimale en maximale waarde van de termen berekend (uiteraard zonder met de wegingscoëfficiënten rekening te houden). Deze waarden kunnen we berekenen aangezien *criterium x* steeds een waarde is tussen 0 en 1.

We merken dat bepaalde criteria overwegen. Om dit te compenseren kunnen we de wegingscoëfficiënten aanpassen. We doen dit op de volgende manier, waarbij a_i de wegingscoëfficiënt van het i-de criterium is:

$$a_1 = \frac{49{,}50}{6{,}64} = 7,46$$

$$a_2 = \frac{49{,}50}{49{,}50} = 1$$

$$a_3 = \frac{49{,}50}{8{,}50} = 5,82$$

$$a_4 = \frac{49{,}50}{5{,}45} = 9,08$$

$$a_5 = \frac{49{,}50}{10{,}86} = 4,56$$

$$a_6 = \frac{49{,}50}{5{,}61} = 8,82$$

In de rest van dit hoofdstuk gebruiken we steeds deze wegingscoëfficiënten. De correctheid van deze coëfficieïten zou nog verder onderzocht kunnen worden. Misschien zijn bepaalde criteria belangrijker dan andere? Wij gaan er van uit dat ze allemaal even belangrijk zijn, maar dit zou nog verder onderzocht moeten worden.

5.3 Verbeteringen in score

In deze sectie willen we evalueren hoe goed het ontworpen tabu search-algoritme werkt. Daalt de score snel? Blijft ze dalen? We bekijken eerst afzonderlijk de melodie en het ritme. Daarna laten we het programma afwisselend ritme en melodie optimaliseren. De tabu tenure bedroeg 1000 voor de 50 en 100 maten en 500 voor 20 maten.

5.3.1 Melodische verbetering

In deze sectie vragen we ons of hoe groot de verbetering van de score is wanneer we enkel de melodie optimaliseren. Geraken we steeds aan een lage waarde? Vinden we snel een lage waarde? We hebben dit eerst onderzocht met een beperkt aantal iteraties (of moves). Nadien hebben we het programma wat langer laten draaien en de resultaten van 500 iteraties geanalyseerd.

50 iteraties

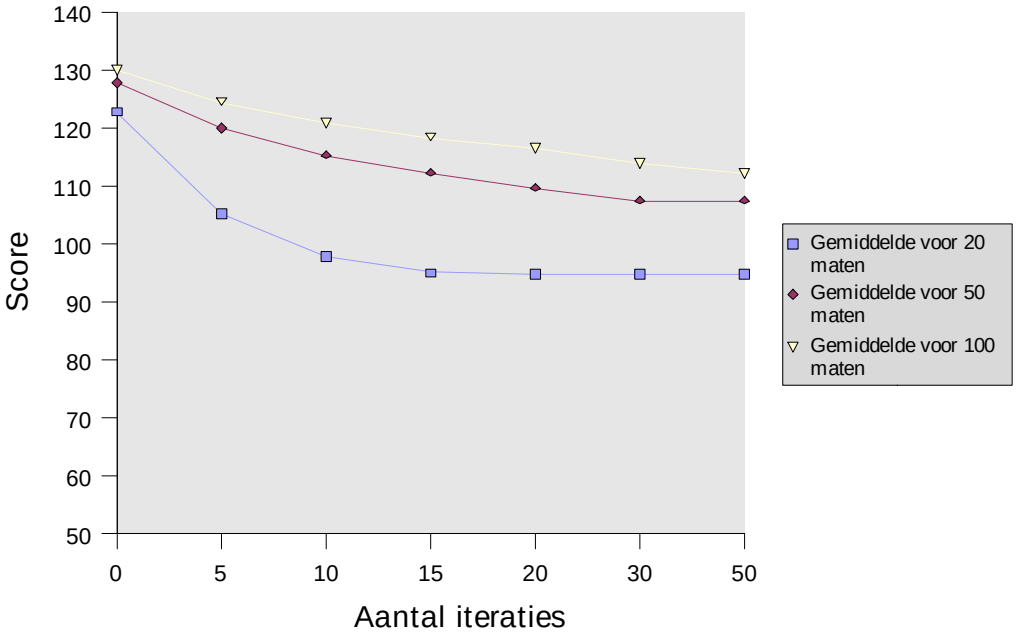
We hebben TabuMusic een aantal keer gerund. Telkens hebben we 50 moves gedaan. Het gemiddelde voor van de testcases is voorgesteld in figuur 5.2. Er is een onderscheid gemaakt tussen muziekstukken van 20, 50 en 100 maten.

Op de figuur is een duidelijk dalende trend te zien. De daling lijkt in het begin sterker te zijn dan op het einde. Verder valt ook op dat kortere muziekfragmenten gemiddeld een lagere beginscore hebben en sneller dalen.

500 iteraties

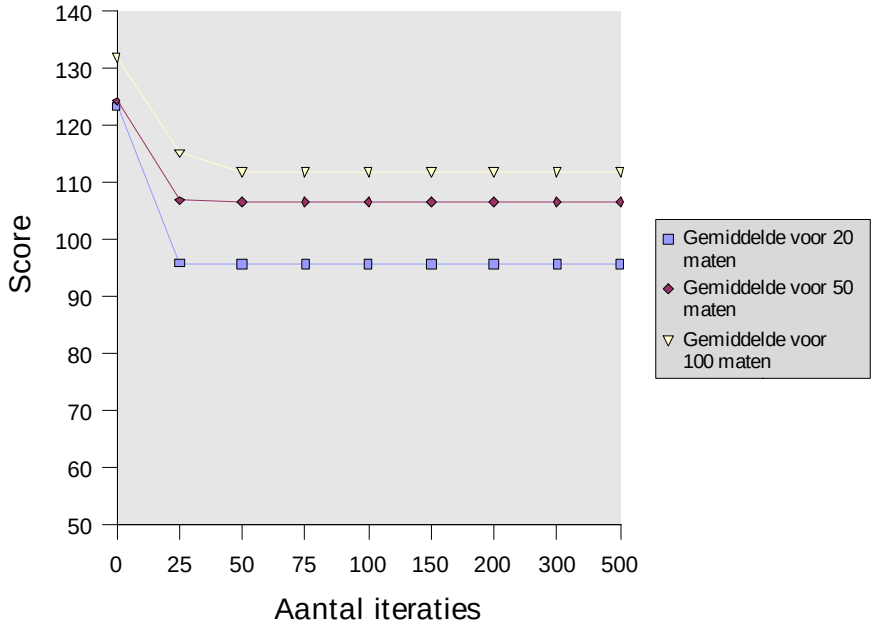
In figuur 5.3 zijn de gemiddelden voorgesteld van enkele test-fragmenten. We hebben TabuMusic nu 500 moves laten maken. Er is weer een onderscheid gemaakt tussen muziekstukken van 20, 50 en 100 maten.

We zien dat er een sterke daling is in het begin. Na ongeveer 25 iteraties worden er geen grote dalingen meer gerealiseerd. Doordat we een zeer grote neighbourhood hebben kan het zijn dat we iets te lokaal blijven zoeken en daarom niet uit het lokale optimum geraken. Daarbij komt nog dat het ritme niet geoptimaliseerd wordt en de



De gemiddelde rekentijd bedroeg:	$<$ 1 sec voor 20 maten
	5 sec voor 50 maten
	19 sec voor 100 maten

Figuur 5.2: Resultaten van de melodische optimalisatie, 50 iteraties



De gemiddelde rekentijd bedroeg:	9 sec voor 20 maten
	48 sec voor 50 maten
	175 sec voor 100 maten

Figuur 5.3: Resultaten van de melodische optimalisatie, 500 iteraties

score dus nooit 0 kan worden. Om uit het lokaal optimum te ontsnappen is er een diversificatie-mechanisme nodig.

5.3.2 Ritmische verbetering

In dit deel vragen we ons af hoe groot de verbetering in score is wanneer we enkel het ritme optimaliseren? Geraken we steeds aan een lage waarde? Geraken we er snel aan? We hebben net zoals bij de melodie een opsplitsing gemaakt tussen 50 en 500 iteraties. De tabu tenure bedroeg 100 voor 50 en 100 maten en 10 voor 20 maten. De lijst is dus minder lang als bij de melodie. Dit is zo gekozen omdat de tabulijst van het ritme iets strenger is, zie sectie 4.6. Ze zal dus sneller moves verbieden. Een voorbeeld maakt dit iets duidelijker:

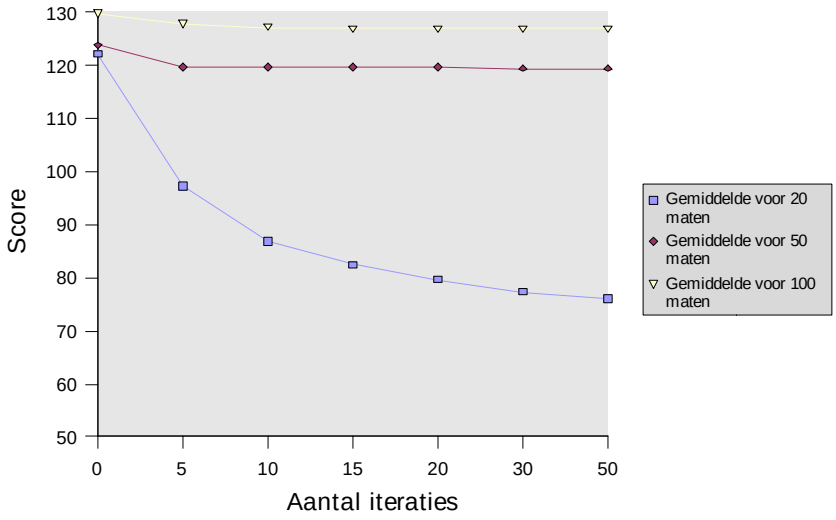
> Op de tabulijst van de melodie kan bijvoorbeeld staan dat noot 2 met als midi-waarde 60 vervangen is. Daartegenover, op de tabulijst van het ritme zal enkel staan dat noot 2 is aangepast en niet dat dit bijvoorbeeld een achtstenoot was. In het geval van de melodie zal men noot 2 nog mogen aanpassen, zolang ze geen midi-waarde 60 krijgt. De ritmeoptimalisatie is iets strenger en zegt dat er niets meer mag veranderen aan noot 2 zolang ze op de tabulijst staat.

Door een kortere tabulijst te nemen voorkomen we dat er na een tijdje geen enkele move meer mogelijk is.

50 iteraties

In figuur 5.4 zijn de resultaten voorgesteld, analoog aan de vorige figuren. Nu is echter het ritme geoptimaliseerd.

Het valt op dat er een sterk dalende trend is bij korte muziekfragmenten. Bij langere stukken is dit echter niet het geval. Het zou kunnen dat er meer dan 50 moves nodig zijn om het ritme van een lang muziekstuk te herschikken. Het zou ook kunnen dat er een zeer snelle convergentie is. Het is daarom nuttig om het programma iets meer moves te laten uitvoeren.



De gemiddelde rekentijd bedroeg:	2 sec voor 20 maten
	7 sec voor 50 maten
	28 sec voor 100 maten

Figuur 5.4: Resultaten van de ritmische optimalisatie, 50 iteraties

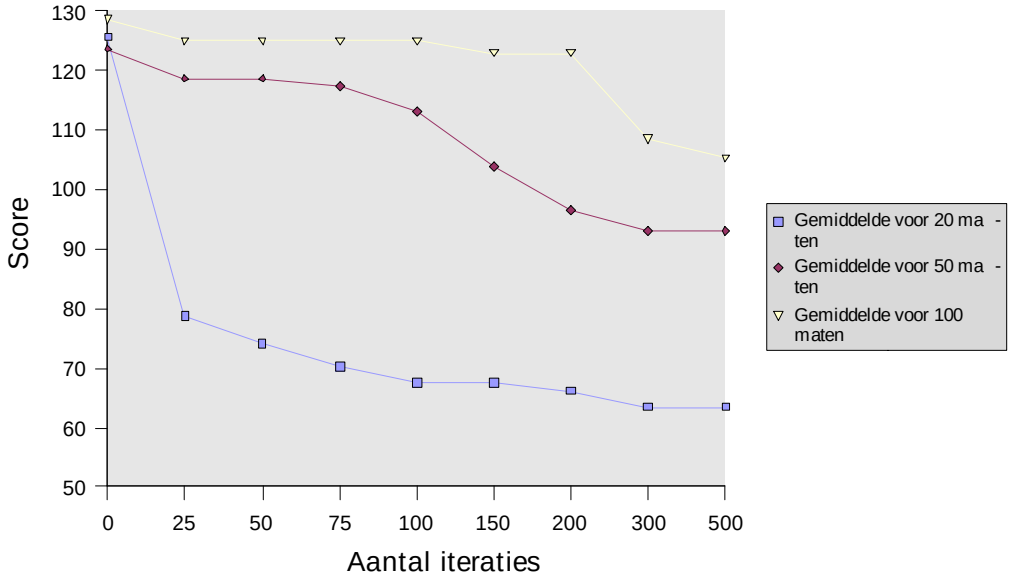
500 iteraties

In figuur 5.5 zijn de resultaten voorgesteld, analoog aan de vorige figuren. Er is nu geoptimaliseerd met 500 iteraties

Er is een continue daling van de doelfunctie. De langere muziekstukken convergeren dus niet zeer snel, ze hebben iets meer tijd nodig om te zakken. Het valt ook op dat er al iets meer rekentijd voor nodig is (meer dan 5 minuten). Kortere fragmenten dalen wel snel en blijven dan nog traag verder dalen. Wanneer we het programma nog veel langer laten lopen komt er misschien wel een punt van convergentie, dit zou nog verder getest kunnen worden. We willen echter niet alleen ritmisch goede muziek, ook de melodie is belangrijk. Daarom gaan we hierna analyseren wat er gebeurd als we beiden optimaliseren.

5.3.3 Ritme en melodie afwisselend verbeterd

Dit deel is eigenlijk het belangrijkste. We vragen ons af hoe groot de verbetering van de score is wanneer we zowel het ritme als de melodie optimaliseren? Geraken we steeds aan een lage waarde? Geraken we er snel aan? We hebben steeds 10 moves het ritme geoptimaliseerd, en dan 10 moves de melodie, enzoverder.



De gemiddelde rekentijd bedroeg:	13 sec voor 20 maten
	134 sec voor 50 maten
	323 sec voor 100 maten

Figuur 5.5: Resultaten van de ritmische optimalisatie, 500 iteraties

50 iteraties

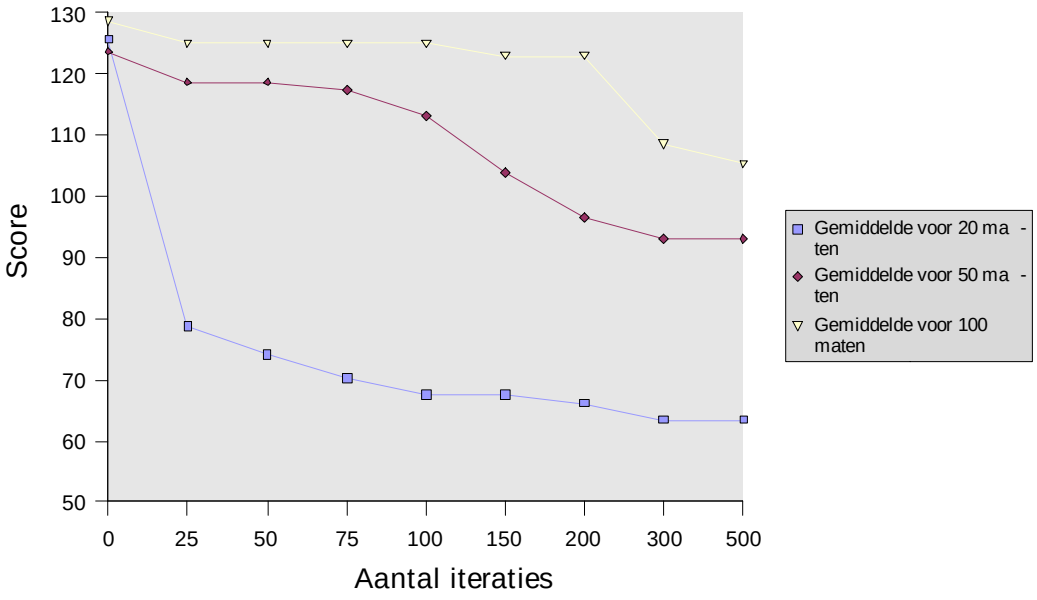
In figuur 5.6 zijn de gemiddelde waarden uitgezet tegen het aantal iteraties. We maken weerom een opsplitsing tussen 20, 50 en 100 maten.

Alle curves dalen. Dit was te verwachten aangezien de melodische en ritmische optimalisatie elk apart ook daalden in het begin. Ook hier zien we, net zoals bij het ritme, dat korte fragmenten sneller dalen.

500 iteraties

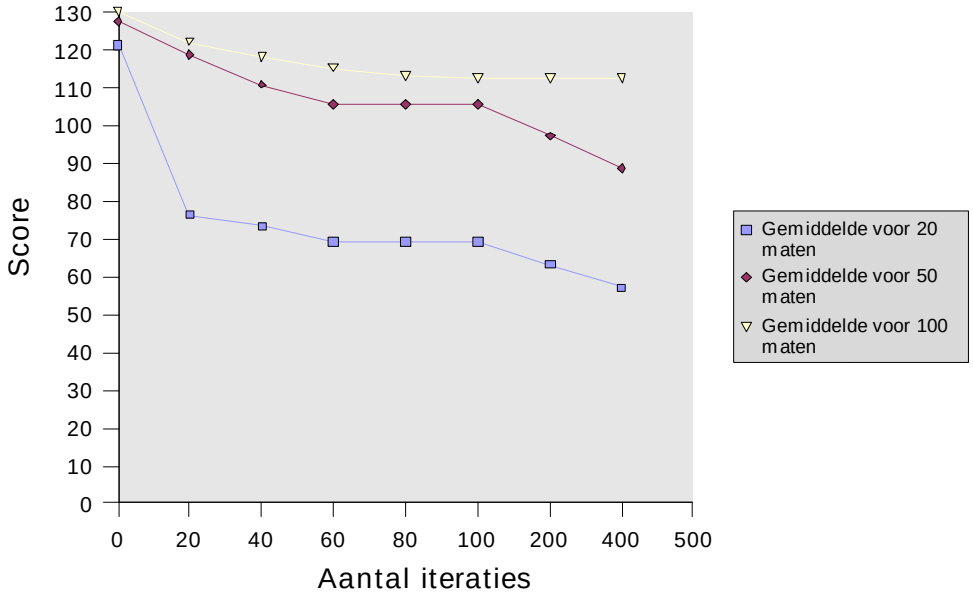
In figuur 5.7 zijn de gemiddelde waarden uitgezet tegen het aantal iteraties. We maken weerom een opsplitsing tussen 20, 50 en 100 maten.

Ook na 500 moves merken we dat de score nog steeds blijft dalen. We herinneren ons dat dit niet het geval was wanneer we enkel de melodie verbeterden. Voor grote fragmenten van 100 maten is de daling zeer zwak. Het zou kunnen dat ze nog meer tijd nodig heeft om sterk te dalen omdat de te verkennen ruimte veel groter is. Eerder zagen we bij het ritme, dat grotere fragmenten meer tijd nodig hadden om een betere score te krijgen. De rekentijd is daarenboven ook langer bij 100 maten



De gemiddelde rekentijd bedroeg:	$<$ 1 sec voor 20 maten
	7 sec voor 20 maten
	23 sec voor 20 maten

Figuur 5.6: Resultaten van de gecombineerde optimalisatie, 50 iteraties



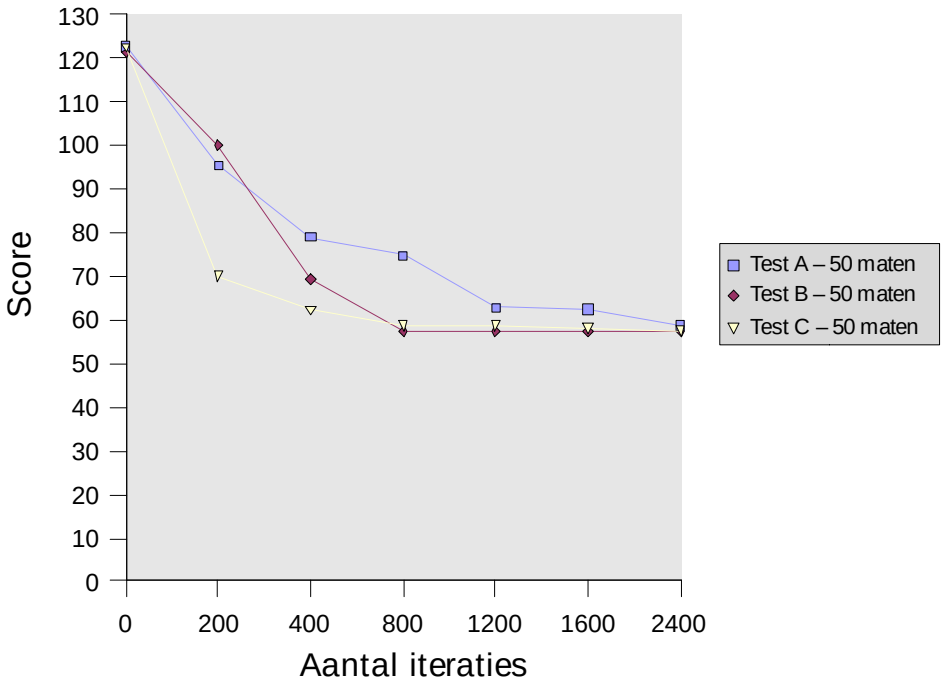
De gemiddelde rekentijd bedroeg:	10 sec voor 20 maten
	46 sec voor 50 maten
	197 sec voor 100 maten

Figuur 5.7: Resultaten van de gecombineerde optimalisatie, 500 iteraties

(ongeveer 3.5 min). Het is nuttig om enkele tests te doen met zeer veel iteraties om te zien of de score van grotere stukken nog blijft dalen.

Nog meer iteraties

Door de lange rekentijd die nodig is om veel iteraties te doen, hebben we de testruns zelf voorgesteld in figuur 5.8 in plaats van het gemiddelde te nemen van meerdere tests.



De gemiddelde rekentijd bedroeg: | 5 min voor 50 maten

Figuur 5.8: Resultaten van de gecombineerde optimalisatie, 2400 iteraties

We merken dat er vooral in het begin een zeer sterk dalende trend is. Nadien is de daling iets minder sterk, test B blijft ongeveer constant. Het kan zijn dat ze na een nog langere tijd verder dalen. Het is echter meer waarschijnlijk dat ze na 2400 moves ongeveer gaan convergeren. Dit kan te wijten zijn aan de aard van de gekozen moves. De neighbourhood is telkens zeer groot en de inividuen uit de neighbourhood verschillen zeer weinig van de originele muziek. Het kan daarom interessant om bijvoorbeeld als move te nemen: wissel twee noten om in plaats van één noot te veranderen.

5.4 Algemene muzikale kwaliteit

Een voorbeeld van een gegenereerd muziekstuk is gegeven in bijlage E. Er bestaat geen twijfel dat de willekeurig gegenereerde muziek echt willekeurig is. Deze muziek is niet aangenaam om naar te luisteren en klinkt zeer 'random'. Het is de persoonlijke mening van de auteur dat geoptimaliseerde muziek, met een lage score, iets beter klinkt dan de willekeurige. Er zitten minder grote intervals in. De overgangen klinken vaak beter en het ritme is consistenter.

De gegenereerde muziek is echter maar zo goed als de doelfunctie toelaat. Er is geen enkele criterium dat iets zegt over de contour, een thema, of dergelijk. Toch lijkt de geoptimaliseerde muziek reeds goed te klinken, ondanks de beperkingen van de doelfunctie.

5.5 Conclusie

Tabu search kan zeker ook worden aangewend op het vlak van computer assisted composing. TabuMusic slaagt er redelijk goed in om de score te minimaliseren. We vinden snel een redelijk lage score. Na een tijdje zijn de dalingen echter niet meer zo groot begint het algoritme te convergeren. Dit kan liggen aan het feit dat de moves zeer beperkt zijn. In verder onderzoek zou men meer en andere moves kunnen inbouwen, om zo meer differentiatie te krijgen.

De muziek die gegenereerd wordt klinkt niet slecht, rekening houdend met de beperkingen van de doelfunctie. Het zou zeer interessant zijn om de doelfunctie verder uit te breiden zodat ze ook toelaat dat er een thema inkomt. We zouden ook de optimale waarden van de verschillende criteria na kunnen gaan. Dit kunnen we doen door de waarde van elk criteria van bestaande bekende fragmenten te berekenen en hiervan het gemiddelde te nemen.

Een andere interessante uitbreiding is het optimaliseren van polyfone muziek. In de hedendaagse klassieke muziek is het zeldzaam dat er maar één noot tegelijk klinkt. Ook zou het interessant zijn om bestaande muziekstukken in te geven en ze te optimaliseren. Zouden ze nog verbeteren?

Besluit

Muziek en computers gaan zeer goed samen. Sinds het begin van de 20ste eeuw is er veel gebeurd op het gebied van computer assisted composing of CAC. Steeds krachtigere computers laten toe om ingewikkeldere programma's te schrijven die beter met muziek om kunnen.

We hebben gezien dat we het componeren van muziek kunnen beschouwen als een optimaliseringsprobleem. Daardoor kunnen we technieken uit de wereld van de optimalisatie gebruiken om het compositieproces te versnellen. Metaheuristieken zijn intelligente zoekstrategieën die hierbij van pas kunnen komen. Genetische algoritmen, een bepaalde soort metaheuristiek, is een veelgebruikte techniek bij het ontwerpen van CAC-programma's. De besproken projecten GenJam en SIEDP, werken allebei op basis van een genetisch algoritme. Zij zijn voorbeelden van een succesvolle toepassing van metaheuristieken voor CAC.

Van een andere metaheuristiek, namelijk tabu search, zijn er weinig of geen voorbeelden bekend. Het leek ons daarom interessant om zelf een toepassing te ontwikkelen (TabuMusic). Het ontworpen programma is bijgeleverd op de cd-rom en is in staat om monofone muziekfragmenten te componeren. Enkele voorbeelden van muziek gegenereerd door TabuMusic staan eveneens op de cd-rom. TabuMusic levert reeds goede resultaten, toch is er na een langere tijd convergentie. Een uitbreiding, die enkele mechanismen voor differentie voorziet, zou hiervoor een oplossing kunnen bieden.

Tabu Search is een zoekstrategie die meer gerichter is dan de veelgebruikte genetische algoritmen. Deze local search strategie komt normalerwijze niet in cirkels vast te zitten door het gebruik van geheugenstructuren zoals de tabulijst. Voor het verkennen van de enorme ruimte van mogelijke muziekstukken is dit zeker handig. We kunnen concluderen dat tabu search een nuttig hulpmiddel kan zijn bij het ontwerpen van CAC-toepassingen.

Bibliografie

Anoniem. Csound. *Website What is CSound Anyway?*, 2004. Beschikbaar online op: `http://www.geocities.com/Paris/3121/csound1.html`.

G. Assayag, C. Rueda, M. Laurson, et al. Computer-assisted composing at ircam: From patchwork to open music. *Computer Music Journal*, 23(3):59–72, 1999.

M. Baroni, S. Maguire, en W. Drabkin. The concept of musical grammar. *Music Analysis*, 2(2):175–208, Juli 1983.

M.L. Besten, T. Stützle, en M. Dorigo. Design of iterated local search algorithms: an example application to the single machine total weighted tardiness problem. *Lecture Notes in Computer Science*, In Proceedings of EvoStim'01:441–452, 2001.

L. Bick. E=mc^2 revisited. *The Bick Report*, 2000. Beschikbaar online op: `http://www.bickfinancial.com/Bick_pdf_Files/BICK%20NewsWin%2000.pdf`.

J.A. Biles. Genjam: a genetic algorithm for generating jazz solos. *In Proceedings of the 1994 International Computer Music Conference, ICMA, San Francisco*, 1994. Beschikbaar online op: `http://www.soi.city.ac.uk/~geraint/papers/STeP98.pdf`.

J.A. Biles. Neural network fitness functions for a musical iga. *In Proceedings of the International ICSC Symposium on Intelligent Industrial Automation (IIA96) and Soft Computing (SOCO96)*, 1996.

J.A. Biles. Autonomous genjam: Eliminating the fitness bottleneck by eliminating fitness. *In Proceedings of the 2001 Genetic and Evolutionary Computation Conference Workshop Program, San Grancisco*, 2001. Beschikbaar online op: `http://www.it.rit.edu/~jab/GECCO01/`.

J.A. Biles. Genjam: Evolutionary computation gets a gig. *Third Conference on Information Technology Curriculum*, 2002. Beschikbaar online op: `http://www.it.rit.edu/~jab/CITC3/Paper.html`.

J.A. Biles. Life with genjam: Interacting with a musical iga. *Website Al Biles*, 2004. Beschikbaar online op: http://www.it.rit.edu/~jab/SMC99/.

E. Bilotta. In search of musical fitness on consonance. *Universitá degli Studi della Calabria*, 2000. Beschikbaar online op: http://galileo.cincom.unical.it/Pubblicazioni/papers/2000/brasile.pdf.

T. Blackburn. Alternate temperaments: Theory and philosophy. *Website Terry Blackburn*, 2004. Beschikbaar online op: http://terryblackburn.us/music/temperament/index.html.

B. Blood. Music history online: Music of the 20th century. *Dolmetsch Online*, 2004. Beschikbaar online op: http://www.dolmetsch.com/theoryintro.htm.

C. Blum en A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, September 2003. Beschikbaar online op: http://webhost.ua.ac.be/eume/MIC2001/MIC2001_451_454.pdf.

B. Boskamp. Compose your own musikalisches würfelspiel. 1997. Beschikbaar online op: http://imagine.xs4all.nl/bram/mozart/.

P. Boulez. *Quoi? quand? comment?* T. Machover, Parijs, 1985.

B.Pardo. Algorithms for chordal analysis. *Computer Music Journal*, 26(2):27–49, 2002.

A.R. Brown. Opportunities for evolutionary music composition. *Australasian Computer Music Association - 10th anniversary conference*, 2002. http://www.media.aau.dk/juko/AS5-Compositional%20Approaches/Brown.pdf.

K.H. Burns. History of electronic and computer music including automatic instruments and composition machines. *Website Dartmouth College*, 2005. Beschikbaar online op: http://eamusic.dartmouth.edu/~wowem/electronmedia/music/eamhistory.html.

CIS. George polya. *Webpagina CIS*, 2004. Beschikbaar online op: http://www.cis.usouthal.edu/misc/polya.html.

D. Corne, M. Dorigo, et al. *New Ideas in Optimization*. Advanced topics in computer science series. McGraw-Hill Publishing Company, Berkshire, 1999.

P. Cox. Math and music: a primer. *Math Mistakes Website*, 2004. Beschikbaar online op: http://members.cox.net/mathmistakes/music.htm.

CSounds. The father of computer music. 2004. Beschikbaar online op: http://www.csounds.com/mathews/.

R. Dannenberg. Project: Nyquist: Summary. *Nyquist Project Website*, 2005. Beschikbaar online op: http://sourceforge.net/projects/nyquist/.

E. Daubresse en G. Assayag. Technology and creation - the creative evolution. *Contemporary Music Review*, 19(2):61–81, 2000.

de Bruijn. Programs for midi. 2004. Beschikbaar online op: http://huizen.dto.tudelft.nl/deBruijn/muziek/midicode.htm.

J. Digalakis en K. Margaritis. Meta-heuristics algorithms. *Universiteit van Macedonië*, 2001. Beschikbaar online op: http://eos.uom.gr/~jason/Ch1.PDF.

Disctronics. Website - glossary. 2004. Beschikbaar online op: http://www.disctronics.co.uk/technology/glossary/glossary_mn.htm.

A. Dittli. The inventor of chess and the emperor. *Website: Web Development*, 2003. Beschikbaar onine op: http://tools.devshed.com/c/a/Web-Development/The-Inventor-of-Chess-and-the-Emperor/.

J. Dréo, A. Ptrowski, et al. *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003.

Encarta. Encarta encyclopedia. 1997. CD-ROM.

FEMS/CAMIL. Granular synthesis. *Florida Electroacoustic Music Studio/Computer Aided Music Instruction Laboratory*, 2004. Beschikbaar online op: http://emu.music.ufl.edu/courses/6445/examples/granola.html.

F. Focacci, F. Laburthe, en A. Lodi. Local search and constraint programming. *MIC'2001 - 4th Metaheuristics International Conference*, 2001. Beschikbaar online op: http://webhost.ua.ac.be/eume/MIC2001/MIC2001_451_454.pdf.

A. Gartland-Jones. Can a genetic algorithm think like a composer. *5th International Conference on Generative Art, 11-13 December*, 2002. Beschikbaar online op: http://www.cogs.susx.ac.uk/users/agj21/drew/cvbiog/papers/ga2002_paper.pdf.

A. Gartland-Jones. Musicblox: a real time algorithmic composition system incorporating a distributed interactive genetic algoritm. *EvoMUSART2003 1st European Workshop on Evolutionary Music and Art*, 2611(1), april 2003. Beschikbaar online op: `http://www.cogs.susx.ac.uk/users/agj21/drew/cvbiog/papers/evomusart_paper.pdf`.

E. Geeurickx. *Functionele Harmonie*. Uitgeverij de Notenboom, 1985.

M. Gendreau. An introductio to tabu search. *Website Universit de Montral*, 2002. Beschikbaar online op: `http://www.ifi.uio.no/infheur/Bakgrunn/Intro_to_TS_Gendreau.htm`.

M. Gendreau, J. Pelleu, en P. Soriano. Improving robustness of a tabu search algorithm for stochastic vehicle routing problems via diversification. *MIC'2001 - 4th Metaheuristics International Conference*, 2001. Beschikbaar online op: `http://webhost.ua.ac.be/eume/MIC2001/MIC2001_435_438.pdf`.

G. Glatt. General midi. 2004. Beschikbaar online op: `http://arts.ucsc.edu/ems/music/equipment/computers/history/history.html`.

J. Glatt. Midi utilities. 1998. Beschikbaar online op: `http://www.borg.com/~jglatt/progs/software.htm`.

F. Glover. Tabu search—part I. *ORSA Journal on Computing*, 1(3):190–206, Zomer 1989. Beschikbaar online op: `http://joc.pubs.informs.org/BackIssues/Vol001/Vol001No03Paper06.pdf`.

F. Glover. Tabu search—part II. *ORSA Journal on Computing*, 2(1):4–32, Winter 1990. Beschikbaar online op: `http://joc.pubs.informs.org/BackIssues/Vol002/Vol002No01Paper01.pdf`.

F. Glover. Tabu search - wellsprings and challenges. *European Journal of Operational Research*, 106:221–225, 1998.

F. Glover. A survey of global optimization methods. *Website Fred Glover*, 2004. Beschikbaar online op: `http://spot.colorado.edu/~glover/`.

M. Harris, A. Smaill, et al. Representing music symbolically. 1993. Beschrikbaar online op: `http://www.soi.city.ac.uk/~geraint/papers/CIM93.pdf`.

J.H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, 1975.

IRCAM. Openmusic. *Forumnet IRCAM*, 2005a. Beschikbaar online op: http://forumnet.ircam.fr/rubrique.php3?id_rubrique=8.

IRCAM. Openmusic. *Website Freesoftware@ircam*, 2005b. Beschikbaar online op: http://freesoftware.ircam.fr//rubrique.php3?id_rubrique=6.

B.J. Jacob. Composing with genetic algorithms. *Proceedings of the International Computer Music Conference*, September 1995. Beschikbaar online op: http://www.ee.umd.edu/~blj/papers/icmc95.pdf.

S. Kirkpatrick, C.D. Gelatt Jr, en M.P. Vecchi. Optimization by simulated annealing. *Science*, 20:671–680, 1983.

L.G.P.M. Kleeven en T.G. Philippi. Muziekcompositie door computers. *Universiteit Utrecht*, 2002. Beschikbaar online op: http://www.phil.uu.nl/onderwijs/cki/cki10/artikelen01-02/kleeven.pdf.

M. Laguna en F. Glover. What is tabu search. *Website Tabusearch.net*, 2004. Beschikbaar online op: http://www.tabusearch.net/Tabu_Search/What_is_Tabu_search.ASP.

A. Land. A recognisable connection between music and math. *Website Math4Future*, 2003. Beschikbaar online op: http://alanland.home.comcast.net/math4furure/inp3.html.

J. Leiber en M. Stoller. Searchin'. 1957.

H.R. Lourenco, O. Martin, en T. Stützle. A beginner's introduction to iterated local search. *MIC'2001 - 4th Metaheuristics International Conference*, 2001. Beschikbaar online op: http://webhost.ua.ac.be/eume/MIC2001/MIC2001_001_006.pdf.

G.F. Luger en W.A. Stubblefield. *Artificial Intelligence, Structures and Srategies for Complex Problem Solving*. Addison-Wesley, derde editie edition, 2003.

B.T. Luke. Simulated annealing cooling scheduls. 2004. Beschikbaar online op: http://fconyx.ncifcrf.gov/~lukeb/simanf1.html.

S. Mansfield. How to interpret abc music notation. *The LeSession pages*, 2004. Beschikbaar online op: http://www.lesession.co.uk/abc/abc_notation.htm.

M. Mastrolilli. Vehicle routing problems. *IDSIA - Istituto Dalle Molle di Studi sull'Intelligenza Artificiale*, 2005. Beschikbaar online op: http://www.idsia.ch/~monaldo/vrp.html#Problem.

MathsoftEngineering en Education. Music and math á la mozart. *Studyworks online*, 2004. Beschikbaar online op: http://www.studyworksonline.com/cda/content/explorations/0,,NAV2-95_SEP1237,00.shtml.

MediaArtNet. Hiller, lejaren; isaacson, leonard - illiac suite: Quartet no. 4 for strings. *Media Art Net*, 2004. Beschikbaar online op: http://www.medienkunstnetz.de/works/illiac-suite/.

MetaheuristicsNetwork. Metaheuristics network website. 2004. Beschikbaar online op: http://www.metaheuristics.org/index.php?main=1.

Z. Michalewicz. *Artificial Intelligence: Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, 1992.

E.R. Miranda. *Composing Music with Computers*. Focal Press, 2001.

NoFreeLunchTheorems. Website no free lunch theorems. 2004. Beschikbaar online op: http://www.no-free-lunch.org/.

I.H. Osman en G. Laporte. Metaheuristics: a bibliography. *Ann. Oper. RES.*, 63:513–623, 1996. Beschikbaar online op: http://www.ifi.uio.no/infheur/Bakgrunn/Intro_to_TS_Gendreau.htm.

G. Papadopoulos en G. Wiggins. A genetic algorithm for the generation of jazz melodies. *Departement of Artificial Intelligence, University of Edinburgh*, 1998. Beschikbaar online op: http://www.soi.city.ac.uk/~geraint/papers/STeP98.pdf.

M. Pearce, D. Meredith, en G. Wiggins. Motivations and methodologies for automation of the compositional process. *City University London, Department of Computing*, 2002. Beschikbaar online op: http://www.titanmusic.com/papers/public/mussci02.pdf.

William Hart Perry Gray et al. A survey of global optimization methods. *Website Sandia National Laboratories*, 2002. Beschikbaar online op: http://www.cs.sandia.gov/opt/survey/.

I. Peterson. Mozart's melody machine. *Science News*, 160(8), 2001. Beschikbaar online op: http://www.sciencenews.org/articles/20010825/mathtrek.asp.

M. Pirlot. General local search methods. *European Journal of Operational Research*, (92):693–511, 1996.

G. Polya. *How to Solve it: A New Aspect of Mathematical Method.* Princeton University Press, 1957.

D. Ponsford, G. Wiggins, et al. Statistical learning of harmonic movement. *Journal of New Music Research*, 1997.

Prins. A simple and effective evolutionary algorithm for the vehical routing problem. *Computers and Operations Research*, 31:1985–2002, 2004.

N. Reynolds. Musical composition and creativity in an ict-enriched learning environment - a case study. *Proceedings of the International Computer Music Conference*, September 1995. Beschikbaar online op: http://www.ee.umd.edu/~blj/papers/icmc95.pdf.

A.M. Rodrigues en J.S. Ferreira. Solving the rural postman problem by memetic algorithms. *MIC'2001 - 4th Metaheuristics International Conference*, 2001. Beschikbaar online op: http://webhost.ua.ac.be/eume/MIC2001/MIC2001_679_684.pdf.

RUN. Radboud university nijmegen, website samsam basic. 2004. Beschikbaar online op: http://www.cmbi.kun.nl/samsam/visualisatie/nacl_gifs.html.

D. Schell. Optimality in musical melodies and harmonic progressions: The travelling musician. *European Journal of Operational Research*, (140):354–372, 2002.

J. Schoenberger. Musical composition with genetic algorithms, with coherency through genotype. 2004.

Selligent. Case study: Ijsboerke klopt op het juiste moment aan de juiste deur dankzij selligents crm. 2005. Beschikbaar online op: http://www.dbs.nl/files/bestanden/success-IJboerke-nl.pdf.

E.A. Silver. An overview of heuristic solution methods. *Website University of Calgary*, 2002. Beschikbaar online op: http://www.haskayne.ucalgary.ca/research/WorkingPapers/research/media/opma_working_papers/2002_15.pdf.

A. Smail, G. Wiggins, en M. Harris. Hierarchical music representation for composition and analysis. *proceedings of the 2nd International Conference on Musical Structures and IT, Marseilles*, 1990.

R. Snarrenberg. Species counterpoint rules. *Website Music Theory III*, 2004. Beschikbaar online op: `http://homepage.mac.com/rsnarren/221/SpeciesRules.pdf`.

Sonhors. Panorama des musiques electroniques. 2005. Beschikbaar online op: `http://sonhors.free.fr/panorama/sonhors2.htm`.

SonyMusicEntertainement. Eras online. *Essentials of Music*, 2001. Beschikbaar online op: `http://www.essentialsofmusic.com/`.

K. Sörensen. Toegepaste optimalisatietechnieken. *Cursus gedoceerd aan de Universiteit Antwerpen, Faculteit Toegepaste Economische Wetenschappen.*

K. Sörensen. Tabu searching for robust solutions. theoretical framework. *Universiteit Antwerpen, Faculteit Toegepaste Economische Wetenschappen*, September 2002. Beschikbaar online op: `http://ideas.repec.org/p/ant/wpaper/2002027.html`.

E. Stephen. History of computer music. *Website CSounds*, 2004. Beschikbaar online op: `http://www.csounds.com/history/`.

tadream.net. History of electronic and computer music including automatic instruments and composition machines. *Tangerine Dream Articles*, 2005. Beschikbaar online op: `http://www.tadream.net/articles/historyofem/history.pdf`.

ThinkQuest. An introduction to computer aided composing. *Website: Computers, Music, and a Little Theory*, 1996. Beschikbaar online op: `http://library.thinkquest.org/3343/web-dov`.

ThinkQuest. A brief history on algorithmic composition. *Thinkquest*, 2004. Beschikbaar online op: `http://library.thinkquest.org/3343/web-docs/history.html`.

M. Towsey, A. Brown, et al. Towards melodic extension using genetic algorithms. *Educational Technology & Society*, 4(2), 2001. Beschikbaar online op: `http://ifets.ieee.org/periodical/vol_2_2001/towsey.pdf`.

C. Truchet, G. Assayag, en P. Codognet. Omclouds, a heuristic solver for musical constraints. *MIC'2003 - 5th Metaheuristics International Conference*, 2003. Beschikbaar online op: `http://143.129.203.3/eume/MIC2003/html/abstracts.php#MIC03_72`.

UCSC. Computer music (so far). *UCSC Electronic Music Web Site*, 2004. Beschikbaar online op: `http://arts.ucsc.edu/ems/music/equipment/computers/history/history.html`.

J. van de Craats en F. Takens. De juiste toon, de juiste stemming. *Nationale Wiskunde Dagen*, 2001. Beschikbaar online op: `http://www.math.leidenuniv.nl/~naw/serie5/deel02/jun2001/pdf/muziek.pdf`.

C. van Nuffelen. *Statistiek I.* Steylaerts, 1998.

A.B. Viana en A.C. de Morais Júnior. Technological improvements in the siedp. *SBC - 2003 Brasili*, 2003. Beschikbaar online op: `http://gsd.ime.usp.br/sbcm/2003/papers/rAlexandre_Viana.pdf`.

J.M. Vidal. Genetic algorithms. 2003. Beschikbaar online op: `http://jmvidal.cse.sc.edu/talks/geneticalgorithms/introduction.xml`.

VRPWeb. Wat is vrp. *Website VRP Web*, 2005. Beschikbaar online op: `http://neo.lcc.uma.es/radi-aeb/WebVRP/VRP-Intro.html`.

C. Walshaw. The abc musical notation language. *Homepage Chris Walshaw*, 2005. Beschikbaar online op: `http://staffweb.cms.gre.ac.uk/~c.walshaw/abc/`.

G. Wiggings, G. Papadopoulos, et al. Evolutionary methods for musical composition. *Website School of Artificial Intelligence*, 1998.

G. Wiggins, M. Harris, en A. Smaill. Representing music for analysis and composition. *Proceedings of the 2nd IJCAI AI/Music Workshop*, pages 63–71, 1989.

G. Wiggins, E. Miranda, et al. Surveying musical representation systems: A framework for evaluations. *Computer Music Journal*, 17(3)(XVII):31–42, 1993a.

G. Wiggins en A. Smail. Musical knowledge: What can artificial intelligence bring to the musician? *Readings in Music and Artificial Intelligence*, 2000. Beschikbaar online op: `http://www.soi.city.ac.uk/~geraint/papers/miranda_collection.pdf`.

G. Wiggins, A. Smaill, en E. Miranda. Music representation - between the musician and the computer. *DAI Research Paper*, 1993b.

Wikipedia. *Wikipedia Encyclopedia*. 2004. Beschikbaar online op: `http://www.wikipedia.org`.

P. Willemé en K. van Rompay. *Onderzoeksmethoden en -technieken*. Universitas, 2001-2002.

J.L. Zychowicz. Western music - a short history. *Classical Music Pages*, 2005. Beschikbaar online op: `http://w3.rz-berlin.mpg.de/cmp/musical_history.html`.

DEEL III

Bijlagen

A Textuele weergave van een MIDI-bestand

Dit voorbeeld is gemaakt met behulp van (Glatt, 1998).

```
MThd | Format=1 | # of Tracks=2 | Division=96

Track #0 ****************************************
    Time       Event
   1: 1:  0 |Tempo  | BPM=60    | micros\quarter=1000000
    |Time Sig  |  4/4      | MIDI-clocks\click=24 | 32nds\quarter=8
    |Key Sig  |  C Major  |
    |End of track|

Track #1 ****************************************
    Time       Event
   1: 1:  0 |On Note  | chan= 1   | pitch=E 4  | vol=80
      3:  0 |Off Note  | chan= 1   | pitch=e 4  | vol=64
    |(Off) Note  | chan= 1   | pitch=c 3
 24 |Off Note  | chan= 1   | pitch=c 3  | vol=64
    |On Note  | chan= 1   | pitch=G 3  | vol=80
      4:  0 |Off Note  | chan= 1   | pitch=g 3  | vol=64
    |On Note  | chan= 1   | pitch=G 3  | vol=80
   2: 1: 72 |Off Note  | chan= 1   | pitch=g 3  | vol=64
    |On Note  | chan= 1   | pitch=F#4  | vol=80
      4: 24 |Off Note  | chan= 1   | pitch=f#4  | vol=64
    |On Note  | chan= 1   | pitch=G 4  | vol=80
   3: 2: 72 |Off Note  | chan= 1   | pitch=g 4  | vol=64
    |On Note  | chan= 1   | pitch=D 3  | vol=80
```

```
   4: 48 |Off Note  | chan= 1   | pitch=d 3  | vol=64
 |On Note  | chan= 1   | pitch=E 3  | vol=80
4: 4:  0 |Off Note  | chan= 1   | pitch=e 3  | vol=64
 |On Note  | chan= 1   | pitch=B 4  | vol=80
5: 2: 48 |Off Note  | chan= 1   | pitch=b 4  | vol=64
 |On Note  | chan= 1   | pitch=F#3  | vol=80
6: 2:  0 |Off Note  | chan= 1   | pitch=f#3  | vol=64
 |On Note  | chan= 1   | pitch=C 4  | vol=80
 |End of track|
```

B Een Orchestra- en een Score-bestand van CSound

Voorbeeld overgenomen van (FEMS/CAMIL, 2004).

Orchestra-bestand

```
;basicgrain.orc

;               70_01_1.orc
;  synthesis:   granular
;               basic granular synthesis
;               uniform pseudorandom selection of
;               grain frequency, inter-grain time,
;               and grain length
;  coded:       6/95  RKP


sr=48000
kr=480
ksmps=100
nchnls=1

instr 1

iamp    = p4    ;amplitude of overall event
ibpch   = p5    ;basepch (center frequency)
```

```
                ibw     = p6    ;bandwidth of random frequency variation
                of grains,
                                ;1 = one semitone variation above and
                                below the base frequency
                imiigt  = p7    ;minimum inter-grain time
                imxigt  = p8    ;maximum inter-grain time
                imigl   = p9    ;minimum grain length
                imxgl   = p10   ;maximum grain length
                iseed   = p11   ;seed value for randh
                igefn   = p12   ;grain envelope function number

                ;inter-grain time: uniform pseudorandom selection
                between imiigt and imxgt
                kr1     randh (imxigt-imiigt)/2,1/imiigt,iseed
                kigt    = imiigt+ (kr1+((imxigt-imiigt)/2))

                ;grain length: uniform pseudorandom selection
                between imigl and imxgl
                kr2     randh (imxgl-imigl)/2,1/imiigt,iseed
                kgl     = imigl+(kr2+((imxgl-imigl)/2))

                ;grain cps random variation: uniform pseudorandom
                selection.
                ;mean = ibpch,  range = (ibpch) + - ibw
                kvar    randh ibw/12,1/imiigt,iseed

                ;envelope for overall event
                kenv    linen iamp,p3*.1,p3,p3*.3

reset:          timout 0,i(kgl)+i(kigt),contin
                reinit  reset
contin:         andx    line  0,i(kgl),4096
                agenv   tablei  andx,igefn
                ioct    = octpch(ibpch)
                icps    = cpsoct(ioct+i(kvar))
                a1      oscili agenv,icps,1
```

```
                out     a1*i(kenv)
endin
```

Score-bestand

```
                ;basicgrain.sco
                ;70_01_1.sco
                ;coded:   RKP  6/95

                ;iamp   amplitude of overall event
                ;ibpch  base pch (center frequency)
                ;ibw    bandwidth of random frequency
                variation of grains,
                        ;1 = one semitone variation above
                        and below  the base frequency
                ;imiigt minimum inter-grain time
                ;imxigt maximum inter-grain time
                ;imigl  minimum grain length
                ;imxgl  maximum grain length
                ;iseed  seed value for randh
                ;igefn  grain envelope function number



;  GEN functions
;  waveform
f1      0       4096    10      1       ;sine function for
grain production

;grain envelope
f31     0       4096    7       0 1024 1 2048 1 1024 0

;score

;ins    strt    dur     iamp    ibpch   ibw     imiigt
```

```
imxigt  imigl   imxgl   iseed   igefn
i1      0       1       10000   9.09    1       .02
.1      .02     .05     .001    31
s
;ins    strt    dur     iamp    ibpch   ibw     imiigt
imxigt  imigl   imxgl   iseed   igefn
i1      0       1       10000   9.09    2       .02
.1      .02     .05     .001    31
s
;ins    strt    dur     iamp    ibpch   ibw     imiigt
imxigt  imigl   imxgl   iseed   igefn
i1      0       1       10000   9.09    3       .02
.1      .02     .05     .001    31
s
;ins    strt    dur     iamp    ibpch   ibw     imiigt
imxigt  imigl   imxgl   iseed   igefn
i1      0       1       10000   9.09    6       .02
.1      .02     .05     .001    31
s
;ins    strt    dur     iamp    ibpch   ibw     imiigt
imxigt  imigl   imxgl   iseed   igefn
i1      0       1       10000   9.09    12      .02
.1      .02     .05     .001    31
e
```

C Voorbeeld van inputmuziek.txt

```
60
20
muziekwillekeurig.abc
muziekgeoptimaliseerd.abc
muzieklog.txt
10
11
100
1
1
1
1
1
1
1
2
1
2
0
```

D Broncode TabuMusic

```
uses crt, sysutils;

const
maxtijden = 1600;
maxmaten = 100;
aantaltonen = 14; {aantaltonen in een toonladder,

wanneer we 2 octaven nemen van do to si}
aantalnoten = 24;
laagstenoot = 60;
maxtabu = 1000;
maxtabu2 = 1000;



type
muztype = array[1..2, 1..maxtijden] of longint;
toontype = array[1..aantaltonen] of longint;
tabutype = array[1..2, 1..maxtabu] of longint;
tabu2type = array[1..maxtabu2] of longint;

var
uur, min, sec, ms : word;
infile, logfile : text;
muziek, muziektijdelijk2 : muztype;
toonladder : toontype;
tabulijst : tabutype;
tabulijst2 : tabu2type;
```

```
aantaltabu, aantaltabu2 : longint;
iters, i, j, k, kies, show, somij, somik, a, b, d, e, f, z, s, y, noot,

 consec, positie, aantaltijden, grondtoon, aantalmaten : longint;
hulp, punten : real;
weging1, weging2, weging3, weging4, weging5, weging6 : real;
bestandsnaam, bestandsnaamin, bestandsnaammuz1, bestandsnaammuz2,

bestandsnaamlog : string;
start, einde, verschil, totaal : TDateTime;
error : boolean;
aspiratiescore : real;
aspiratie : muztype;

function doelfunctie(muziekd : muztype) : real;
var
crit5, crit1, crit8, crit10, score1, crit7, crit2, diss, crit3,

 crit11, crit4, crit6 : real;
aantal: longint; {aantal verschillende tonen}
max, min, herhaald, c, aantalntoon, somb,geenrust, aantalrust, hoogste,

laagste, duren, ok, b, aantalzrust  : longint;
intervals, stijgend, verschil : longint;

{stijgend = aantal stijgende intervallen}
noten:longint; {aantal noten waaruit het stuk bestaat}
notenlijst : array[1..aantalnoten] of longint;
muzrust, muziekr, muziek2 : muztype;
duur : array[1..16] of longint;
doel1, doel2, doel3, doel4, doel5, doel6 : real;

begin
{eerste criterium: pitch variety= aantal verschillende noten

(=aantal)/aantalnoten(=noten)}
```

```
{1. de verschillende noten tellen}
{lijst leegmaken}
for i:=1 to aantalnoten do
    notenlijst[i] := 0;

{notenlijst = 1 als de noot erin zit}
for i:=1 to aantaltijden do
    begin
    k := (muziekd[1,i]-laagstenoot+1);
    if (k <> (-laagstenoot))
         then notenlijst[k] := 1; {rusten(-1) niet meetellen}
   end;


{aantal enen tellen in notenlijst}
aantal := 0;
for i:=1 to aantalnoten do
    begin
    if notenlijst[i] = 1
    then aantal := aantal +1;
    end;


{Uit hoeveel noten bestaat het stuk?}
noten := 0;
for i:=1 to aantaltijden do
    begin
    if (muziekd[2,i] = 0) and (muziekd[1,i]<>(-1))
       then noten := noten +1;
   end;


crit1:=(aantal/noten);


{criterium 2: dissonant intervals.  Een interval van 10 krijgt een
```

```
 dissonantiescore (=diss) van 0.5. Intervallen van 6, 11 of >= 13

krijgen een diss van 1.0.
De dissonantiescore wordt gedeeld door het aantal intervals}

{allenoten in een matrix zetten zonder de rusten en lange noten: muzrust}
aantalzrust := 0;
for i:=1 to aantaltijden do
    begin
   if (muziekd[1,i] <> -1) and (muziekd[2,i] = 0) then
       begin
       aantalzrust := aantalzrust+1;
       muzrust[1,aantalzrust] := muziekd[1,i];
       muzrust[2,aantalzrust] := muziekd[2,i];
       end;
    end;


{bepalen dissonantiescore}

diss:=0;

for i:=1 to (aantalzrust-1) do
    begin
    verschil := ABS(muzrust[1,i]-muzrust[1,i+1]);
    if (verschil=10) then begin diss := diss + 0.5; end;
    if (verschil=6)  then begin diss := diss +1; end;
    if (verschil=11) then begin diss := diss +1; end;
    if (verschil>=13) then begin diss := diss +1; end;
   end;
if noten = 1 then crit2 := 1
    else crit2 := diss/(noten-1);


{criterium 3: contour direction. De som van alle stijgende intervals

gedeeld door de som van alle intervals}
```

```
intervals:=0;
stijgend := 0;

for i:=1 to (aantalzrust-1) do
    begin
    if muzrust[1,i+1] > muzrust[1,i]
       then stijgend := stijgend + muzrust[1,i+1]-muzrust[1,i];
    intervals := intervals + ABS(muzrust[1,i+1] - muzrust[1,i]);
   end;
if intervals = 0 then crit3 := 1
  else crit3 := stijgend/(intervals);


{criterium 4: contour stability. aantal opeenvolgende

intervallen die in dezelfde richting evolueren.  }
{tellen aantal opeenvolgende intervals in dezelfde richting}
consec := 0;
for i:=1 to (aantalzrust-2) do
    begin
    if (muzrust[1,i+2] >= muzrust[1,i+1]) and

(muzrust[1,i+1] >= muzrust[1,i])
       then consec := consec + 1;
    if (muzrust[1,i+2] <= muzrust[1,i+1]) and

(muzrust[1,i+1] <= muzrust[1,i])
       then consec := consec + 1;
    end;
if noten=2 then crit4 := 1
else crit4 := consec/(noten-2);


{criterium 5: rythmic variety.
aantal verschillende nootduraties gebruikt / 16

(aantal verschillende nootduraties ongeveer mogelijk}
```

```
for i:=1 to 16 do
    duur[i] := 0;

i:=1;
while i<= aantaltijden do

   begin
    b := 0;
    if muziekd[2,i] = 1 then
         begin
                while (muziekd[2,i+b] = 1) do
                    begin
                {gewoon tellen}
                    b := b +1;
                      end;

         end;

    if muziekd[2,i+b] = 0 then
         begin
                somb := b+1;
                duur[somb] := duur[somb]+1;
        end;

    i := i+b+1;

end;

{aantal verschillende tijdsduren tellen}
duren := 0;
for i := 1 to 16 do
    begin
    if duur[i] <> 0
       then duren := duren + 1;
   end;
```

```
crit5 := duren/16;


{Criterium 6: rythmic change
(Max noot duratie + min noot duratie) / 16}
{zoek max noot duratie: max}
max := 1;
min := 1;
ok := 0;
i:=16;
while ok = 0 do
    begin
    if duur[i] <> 0  then
       begin
       max := i;
       ok := 1;
      end;
    i := i-1;
    end;

{vind de minimum duur}
ok := 0;
i := 1;
while ok = 0 do
    begin
    if duur[i] <> 0 then
       begin
       min := i;
       ok := 1;
       end;
    i := i+1
    end;


crit6 := (max-min)/16;
```

```
doel1 := (ABS(crit1-0.27))/0.11;
doel2 := (ABS(crit2-0.01))/0.02;
doel3 := (ABS(crit3-0.49))/0.06;
doel4 := (ABS(crit4-0.40))/0.11;
doel5 := (ABS(crit5-0.24))/0.07;
doel6 := (ABS(crit6-0.32))/0.11;


{totale score}
doelfunctie := (weging1*doel1)+(weging2*doel2)+(weging3*doel3)+

(weging4*doel4)+(weging5*doel5)+(weging6*doel6);
if show = 1 then
  begin
  writeln(logfile, 'Pitch variety: ', #9,#9, doel1:0:6);
  writeln(logfile, 'Dissonant intervals:', #9,  doel2:0:6);
  writeln(logfile, 'Contour direction: ', #9,  doel3:0:6);
  writeln(logfile, 'Contour stability: ', #9, doel4:0:6);
  writeln(logfile, 'Rhythmic variety: ', #9,  doel5:0:6);
  writeln(logfile, 'Rhythmic range: ',  #9, doel6:0:6);
  end;

end;



procedure schrijfabc(muziekschrijf: muztype ;

bestandsnaamschrijf : string);
var
outfile : text;
teller, tijd : longint;
abc : string;

begin
assign(outfile, bestandsnaamschrijf);
```

```
rewrite(outfile);
writeln(outfile,'X:1');
writeln(outfile,'T:Muziekfragment TabuMusic');
writeln(outfile,'M:4/4');
writeln(outfile,'K:C');
writeln(outfile,'L:1/16');

tijd := 0;
teller := 1;
for i:=1 to aantaltijden do
begin
{als 2e lijn = 0 dan schrijven}
if muziekschrijf[2,i] = 0 then
    begin
    {noot schrijven naar bestand}
    if muziekschrijf[1,i] = 60 then write(outfile,'C',teller,' ');
    if muziekschrijf[1,i] = 61 then write(outfile,'^C',teller,' ');
    if muziekschrijf[1,i] = 62 then write(outfile,'D',teller,' ');
    if muziekschrijf[1,i] = 63 then write(outfile,'_E',teller,' ');
    if muziekschrijf[1,i] = 64 then write(outfile,'E',teller,' ');
    if muziekschrijf[1,i] = 65 then write(outfile,'F',teller,' ');
    if muziekschrijf[1,i] = 66 then write(outfile,'^F',teller,' ');
    if muziekschrijf[1,i] = 67 then write(outfile,'G',teller,' ');
    if muziekschrijf[1,i] = 68 then write(outfile,'^G',teller,' ');
    if muziekschrijf[1,i] = 69 then write(outfile,'A',teller,' ');
    if muziekschrijf[1,i] = 70 then write(outfile,'_B',teller,' ');
    if muziekschrijf[1,i] = 71 then write(outfile,'B',teller,' ');
   if muziekschrijf[1,i] = 72 then write(outfile,'c',teller,' ');
    if muziekschrijf[1,i] = 73 then write(outfile,'^c',teller,' ');
    if muziekschrijf[1,i] = 74 then write(outfile,'d',teller,' ');
    if muziekschrijf[1,i] = 75 then write(outfile,'^d',teller,' ');
    if muziekschrijf[1,i] = 76 then write(outfile,'e',teller,' ');
    if muziekschrijf[1,i] = 77 then write(outfile,'f',teller,' ');
    if muziekschrijf[1,i] = 78 then write(outfile,'^f',teller,' ');
    if muziekschrijf[1,i] = 79 then write(outfile,'g',teller,' ');
    if muziekschrijf[1,i] = 80 then write(outfile,'^g',teller,' ');
```

```
    if muziekschrijf[1,i] = 81 then write(outfile,'a',teller,' ');
    if muziekschrijf[1,i] = 82 then write(outfile,'_b',teller,' ');
    if muziekschrijf[1,i] = 83 then write(outfile,'b',teller,' ');
    if muziekschrijf[1,i] = -1 then write(outfile,'z',teller,' ');

    teller := 1; {we beginnen terug met een nieuwe noot die

 min 1 tel duurt}

    tijd := tijd + 1;

    if tijd = 12 then
       begin
       writeln(outfile,'');writeln(outfile,'');
       tijd := 0;
       end;
    end;

if muziekschrijf[2,i] = 1 then
    begin
    teller := teller + 1;
    end;

end;
close(outfile);


end;



function noottransponeren(muziektransp: muztype;

scoretransp: real): muztype;

var
muziekmeer, muziektijdelijk : muztype;
```

```
scoretijdelijk, scoremeer, beginscore : real;
ok1, x , v, dalen, gelijk, tabuxmeer, tabuymeer: longint;
vervangen : boolean;
tabux, tabuy : longint;

begin
tabux := 0;
tabuy := 0;
noottransponeren := muziektransp;
muziektijdelijk := muziektransp;
dalen := 1;
scoremeer := 10000;
beginscore := scoretransp;

for x := 1 to aantaltonen do
begin
{probeer elke noot te vervangen}
y := 1;
while y <= aantaltijden do
begin
    {vervang 1 noot}
    z := 0;
   ok1 := 0;
    while ok1 = 0 do
       begin
      muziektijdelijk[1,y+z] := toonladder[x];

       if muziektijdelijk[2,y+z]= 0 then
         begin
          ok1 := 1;
          end;
      z := z+1;
       end;
    scoretijdelijk := doelfunctie(muziektijdelijk);
   vervangen := false;
```

```
    {geen betere gevonden}
   {best mogelijke slechte oplossing onthouden}
    if scoretijdelijk > beginscore then
begin
if scoremeer > scoretijdelijk then
begin
scoremeer := scoretijdelijk;
muziekmeer := muziektijdelijk;
                                tabuxmeer := y;
                                tabuymeer := muziektijdelijk[1,y];
end;
end;

    {betere gevonden!}

   {is het dezelfde oplossing?}
    gelijk := 1;
   for v := 1 to aantaltijden do
       begin
       if muziektijdelijk[1,v] <> muziektransp[1,v] then
          gelijk := 0;
       end;
   for v := 1 to aantaltijden do
       begin
      if muziektijdelijk[2,v] <> muziektransp[2,v] then
          gelijk := 0;
      end;


   if (scoretijdelijk <= scoretransp) and (gelijk = 0) then
       begin
        vervangen := true;
      for v:=1 to aantaltabu do
       begin
         if (muziektijdelijk[1,y]=tabulijst[2,v]) then
             begin
```

```
              if (tabulijst[1,v]=y) then
                begin
                 vervangen:= false;
                 end;
              end;
           end;

{toch vervangen als het aspiratieniveau is bereikt}
if scoretijdelijk < aspiratiescore then
begin
aspiratie := muziektijdelijk;
aspiratiescore := scoretijdelijk;
vervangen := true;
end;

{geen TABU}
       if vervangen = true then
          begin
         {noot vervangen}
  dalen := 0;
          scoretransp := scoretijdelijk;
         noottransponeren := muziektijdelijk;
{onthouden welke getallen er in de tabulijst moeten komen}
tabux := y;
tabuy := muziektijdelijk[1,y];
         end;
       end;

     muziektijdelijk := muziektransp;
    y := y+z;
    end;
end;

if dalen = 1 then
    begin
    scoretransp := scoremeer;
```

```
   noottransponeren := muziekmeer;
        tabux := tabuxmeer;
        tabuy := tabuymeer;
    end;


{tabulijst bijvullen als het is aangenomen}
if (dalen = 0) or (dalen=1) then
begin
{tabulijst opschuiven}
for v := (aantaltabu-1) downto 1 do
         begin
        tabulijst[1,v+1] := tabulijst[1,v];
         tabulijst[2,v+1] := tabulijst[2,v];
end;

{tabulijst bijvullen}
tabulijst[1,1] := tabux;
tabulijst[2,1] := tabuy;

end;
end;



function ritmeverbeteren(muziekrverb: muztype; punten2:real): muztype;
var
ritmeverb, muziekmeer2 : muztype;
telnoten: longint;
c1, rest, x2, y1, v2, u, dalen2, gelijk2: longint;
ritmemuziek, ritmemuziek2 : array[1..2, 1..maxtijden+1] of longint;
vervangen2, ok2 : boolean;
scoretijdelijk2, scoremeer2, beginscore2 : real;
tabux1, tabux1meer : longint;

begin
```

```
ritmeverb := muziekrverb;
muziektijdelijk2 := muziekrverb;
ritmeverbeteren := muziekrverb;
dalen2 := 1;
scoremeer2 := 10000;
beginscore2 := punten2;
tabux1 := 0;


{noten tellen}
telnoten := 0;
for x2 := 1 to aantaltijden do
begin
if muziekrverb[2,x2] = 0 then
begin
    telnoten := telnoten + 1;
    end;
end;


{ritmemuziek initialiseren}
for x2 := 1 to 2 do
begin
for y1 := 1 to (maxtijden+1) do
ritmemuziek[x2,y1] := 0;
end;


{in speciale matrix zetten. Rij : midiwaarde en Kolom :

 aantaltijden dat de noot duurt}

v2 := 0;
u := 1;
for x2 := 1 to aantaltijden do
begin
    if muziekrverb[2,x2] = 1 then
```

```
    begin
       v2 := v2 +1;
   end;

    if muziekrverb[2,x2] = 0 then
    begin
      v2 := v2 + 1;
       ritmemuziek[1,u] := muziekrverb[1,x2];
       ritmemuziek[2,u] := v2;

       u := u + 1;
       v2 := 0;
    end;


end;


{probeer het ritme van de noot te vervangen door elk mogelijk ander}
{eerst voor de vde noot noot}
for v2 := 1 to telnoten do
begin

for u := 1 to 16 do
begin
   ritmemuziek2 := ritmemuziek;
   ritmemuziek2[2,v2] := u;

    {terug omzetten in goeie matrix}

    y1 := 1;
    c1 := 1;
    rest := ritmemuziek2[2,c1];     {rest: aantal tijden

dat de noot nog verderklinkt}
   while y1 <= aantaltijden do
       begin
```

```
 ok2 := true;
 {per kolom van ritmemuziek: }

    {begin}
    muziektijdelijk2[1,y1] := ritmemuziek2[1,c1];
    {als er geen noten meer zijn: }
    if ritmemuziek2[2,c1] = 0 then
       begin

       muziektijdelijk2[1,y1] := ritmemuziek[1,v2];
       muziektijdelijk2[2,y1] := 1;
       rest := 0;
       end;

    if rest = 1 then
       begin
       muziektijdelijk2[2,y1] := 0;
       c1 := c1 + 1;
       rest := ritmemuziek2[2,c1];
       ok2 := false;
       end;

    if (rest > 1) and (ok2) then
       begin
       muziektijdelijk2[2,y1] := 1;
       rest := rest - 1;
       end;

y1 := y1 + 1;
 end;


 {laatste noot moet stoppen: }
muziektijdelijk2[2,aantaltijden] := 0;
```

```
    {doelfunctie berekenen}
    scoretijdelijk2 := doelfunctie(muziektijdelijk2);
                vervangen2 := false;

{geen betere gevonden}
{best mogelijke oplossing onthouden}
if scoretijdelijk2 > beginscore2 then
begin
if scoremeer2 > scoretijdelijk2 then
begin
  scoremeer2 := scoretijdelijk2;
muziekmeer2 := muziektijdelijk2;
                                  tabux1meer := v2;
end;
end;
  {is het dezelfde oplossing?}
   gelijk2 := 1;
    for x2 := 1 to aantaltijden do
       begin
      if muziektijdelijk2[1,x2] <> muziekrverb[1,x2] then
          gelijk2 := 0;
       end;
   for x2 := 1 to aantaltijden do
       begin
       if muziektijdelijk2[2,x2] <> muziekrverb[2,x2] then
          gelijk2 := 0;
       end;


   {lage score}
    if (scoretijdelijk2 <= punten2) and (gelijk2 = 0) then
       begin
       vervangen2 := true;
        for x2 := 1 to aantaltabu2 do
          begin
```

```
          if (tabulijst2[x2]=v2) then
             begin
             vervangen2 := false;
             end;
          end;
       end;

{toch vervangen als het aspiratieniveau is bereikt}
if scoretijdelijk2 < aspiratiescore then
begin
aspiratie := muziektijdelijk2;
aspiratiescore := scoretijdelijk2;
vervangen2 := true;
end;
    {hogere score}
   if vervangen2 = true then
       begin
       {nieuwe oplossing aannemen}
dalen2 := 0;
       punten2 :=  scoretijdelijk2;
       ritmeverbeteren := muziektijdelijk2;
{onthouden getallen tabulijst}
tabux1 := v2;

       end;
end;
end;

if dalen2 = 1 then
begin
punten2 := scoremeer2;
ritmeverbeteren := muziekmeer2;
                tabux1 := tabux1meer;
end;

        if (dalen2 = 0) or (dalen2=1) then
```

```
begin
{tabulijst}
       for x2 := (aantaltabu2-1) downto 1 do
          begin
          {tabulijst opschuiven}
          tabulijst2[x2+1] := tabulijst2[x2];
          end;
       {tabulijst bijvullen}
       tabulijst2[1] := tabux1;
       {tabulijst drukken}
end;

end;


begin
{scherminstellingen}
clrscr;
textbackground(white);
textcolor(black);


{tabulijst initialiseren}
for i := 1 to 2 do
   begin
   for j := 1 to aantaltabu do
      tabulijst[i,j] := 0;
   end;

for i := 1 to aantaltabu2 do
   begin
   tabulijst2[i] := 0;
   end;

{initialisatie}
error := true;
```

```
{init infile}
bestandsnaamin := 'inputmuziek.txt';
assign (infile, bestandsnaamin);
reset(infile);
readln(infile, grondtoon);
readln(infile, aantalmaten);
readln(infile, bestandsnaammuz1);
readln(infile, bestandsnaammuz2);
readln(infile, bestandsnaamlog);
readln(infile, iters);
readln(infile, aantaltabu);
readln(infile, aantaltabu2);
readln(infile, weging1);
readln(infile, weging2);
readln(infile, weging3);
readln(infile, weging4);
readln(infile, weging5);
readln(infile, weging6);

{init logfile}
assign(logfile, bestandsnaamlog);
rewrite(logfile);

totaal := 0;


if (grondtoon <60) or (grondtoon>71) then
    begin
    writeln('U hebt een verkeerde grondtoon opgegeven!');
    error := false;
    end;


if (aantalmaten > maxmaten) then
    begin
```

```
    writeln('U hebt een te groot aantal maten opgegeven!');
    error := false;
   end;


{vul toonladderverzameling}
{zoek de laagste noot van de toonladder}
noot := grondtoon-12;
positie := 1;
while noot < laagstenoot do
    begin
    if (positie = 1) or (positie = 2) or (positie = 4)

or (positie = 5)  or (positie = 6)
       then noot := noot + 2
       else noot := noot + 1;
    positie := positie +1;
    if positie = 8 then positie := 1;
    end;
    toonladder[1] := noot;

{genereer de rest van de toonladder}
for i:=2 to aantaltonen do
    begin
    if (positie = 1) or (positie = 2) or (positie = 4)

or (positie = 5)  or (positie = 6)
       then noot := noot + 2
       else noot := noot + 1;
    positie := positie +1;
    if positie = 8 then positie := 1;
    toonladder[i] := noot;
    end;


{genereer een eerste willekeurige oplossing}
aantaltijden := aantalmaten * 16;
```

```
randomize;
i := 1;
while i<= aantaltijden do
    begin
    a := random(aantaltonen+1); {random toon}

    k := random(16)+1;          {random duur}

    if ((i+k) > aantaltijden) then
       begin
      k := aantaltijden - i+1;{-k+1}
       end;


       {eerste rij opvullen}
       for j := 1 to k do
          begin
          if a=0 then
             begin
             somij := i+j-1;
             muziek[1,somij] := -1;
             end;

          if a<> 0 then
             begin
             somij := i+j-1;
            muziek[1,somij] := toonladder[a];
             end;
          end;

      {tweede rij opvullen}
       if k > 1 then
          begin
          for j := 1 to (k-1) do
             begin
             somij := i+j-1;
```

```
            muziek[2,somij] := 1
            end;
         end;

       somik := i+k-1;
       muziek[2,somik] := 0;

      i := i + k;
       if k = 0 then i := aantaltijden+1;

    end;


{laatste element tweede rij moet nul zijn}
muziek[2,aantaltijden] := 0;

{wegschrijven willekeurig muziekstuk}
schrijfabc(muziek, bestandsnaammuz1);
writeln(logfile, '*******************************');
writeln(logfile, '*      Logfile TabuMusic      *');
writeln(logfile, '*******************************');
writeln(logfile);
writeln(logfile);
writeln(logfile, 'De willekeurig gegenereerde muziek is

opgeslagen als ', bestandsnaammuz1);
writeln(logfile);
writeln(logfile, 'Grondtoon: ', #9, grondtoon);
writeln(logfile, 'Aantal maten: ', #9, aantalmaten);
writeln(logfile);


{berekenen score willekeurig muziekstuk}
show := 0;
punten := doelfunctie(muziek);
{aspiratieniveau initialiseren}
for i :=1 to 2 do
```

```
    begin
    for j := 1 to aantaltijden do
       begin
       aspiratie[i,j] := muziek[i,j];
       end;
    end;

aspiratiescore := punten;

writeln('********************');
writeln('*    TabuMusic    *');
writeln('********************');
writeln;
writeln(logfile, 'De score van het willekeurig muziekstuk

bedraagt: ', punten:0:6);
writeln(logfile);
show := 1;
punten := doelfunctie(muziek);
show := 0;
writeln(logfile);
writeln(logfile, '-----------------------------------');
writeln(logfile);
writeln(logfile, 'Er werd geoptimaliseerd met steeds ',iters,

' iteraties: ');
writeln(logfile, 'Lengte tabulijst melodie: ', #9,aantaltabu);
writeln(logfile, 'Lengte tabulijst ritme: ',  #9,aantaltabu2);
writeln(logfile);
writeln(logfile, '-----------------------------------');
writeln(logfile);
writeln(logfile, 'WEGINGSCOEFFICIENTEN:');
writeln(logfile);
writeln(logfile, 'Wegingscoefficient Pitch variety: ', #9,

weging1:0:3);
writeln(logfile, 'Wegingscoefficient Dissonant intervals:', #9,
```

```
weging2:0:3);
writeln(logfile, 'Wegingscoefficient Contour direction: ', #9,

weging3:0:3);
writeln(logfile, 'Wegingscoefficient Contour stability: ',  #9,

weging4:0:3);
writeln(logfile, 'Wegingscoefficient Rhythmic variety: ',  #9,

weging5:0:3);
writeln(logfile, 'Wegingscoefficient Rhythmic range: ', #9,

 weging6:0:3);
writeln(logfile);
writeln(logfile, '-----------------------------------');
writeln(logfile);
writeln(logfile, 'OPTIMALISATIE: ');
writeln(logfile);



writeln('Beginscore: ',punten:0:6);
write('Bezig met verwerken...');

s := 1;

while (s = 1) and (error) do


begin
show := 0;

readln(infile, kies);

if kies = 1 then
begin
```

```
writeln(logfile, 'MELODIE: ');
writeln(logfile);
{STARTTIJD}
start := Time;
{tabu search melodie}
show := 0;
for d := 1 to iters do
    begin
    {tabu search: verbeteren toonhoogte noten}
    muziek := noottransponeren(muziek, punten);
    if d = iters then show := 0;
    punten := doelfunctie(muziek);
    show := 0;
   end;

    write('.');
   einde := Time;
    verschil := einde - start;
    DeCodeTime(verschil, uur, min, sec, ms);
   totaal := totaal + verschil;
    writeln(logfile, 'Score: ', punten:0:10);
    writeln(logfile, 'Beste score: ', aspiratiescore:0:10);
    writeln(logfile, 'Rekentijd: ', uur, ' uur, ', min, ' min, ',

 sec, ' sec, ', ms, ' ms');
    writeln(logfile);
    writeln(logfile);

end;


if kies = 2 then
begin
{tabu search ritme}
writeln(logfile, 'RITME: ');
writeln(logfile);
show := 0;
```

```
hulp := punten;
start := Time;
for d := 1 to iters do
    begin
    muziek := ritmeverbeteren(muziek, punten);
    if d = iters then show := 0;
    punten := doelfunctie(muziek);
    show := 0;

    end;
    write('.');
    einde := Time;
    verschil := einde - start;
    DeCodeTime(verschil, uur, min, sec, ms);
    totaal := totaal + verschil;
writeln(logfile, 'Score: ', punten:0:10);
    writeln(logfile, 'Beste score: ', aspiratiescore:0:10);
    writeln(logfile, 'Rekentijd: ', uur, ' uur, ', min, ' min, ',

 sec, ' sec, ', ms, ' ms');
    writeln(logfile);
    writeln(logfile);
end;


if (kies <> 1) and (kies <> 2) then
begin
s := 0;
end;

end;

writeln;
writeln;
writeln('De beste score is: ',aspiratiescore:0:6);
writeln('De output is geschreven naar ',bestandsnaamlog);
writeln(logfile, '-----------------------------------');
```

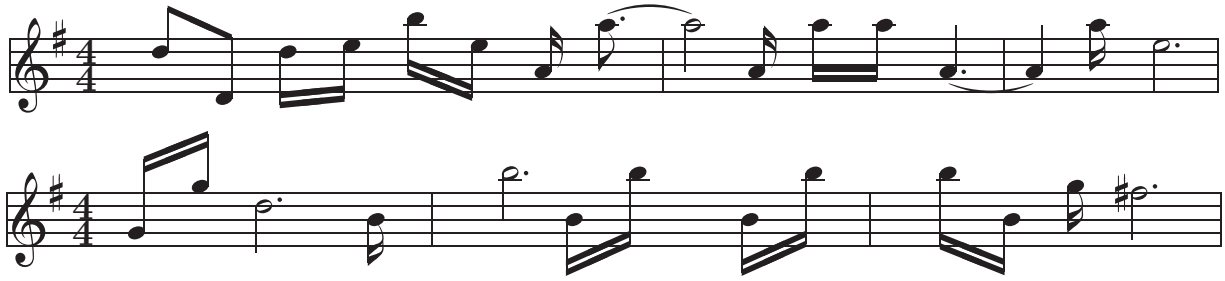
```
writeln(logfile, '-----------------------------------');
writeln(logfile);
writeln(logfile, 'Het GEOPTIMALISEERDE muziekfragment is opgeslagen

onder ', bestandsnaammuz2);
 DeCodeTime(totaal, uur, min, sec, ms);
writeln(logfile, 'Totale rekentijd: ',  uur, ' uur, ', min, ' min, ',

sec, ' sec, ', ms, ' ms');
writeln(logfile);
writeln(logfile, 'EINDSCORE:');
writeln(logfile);
show := 1;
punten := doelfunctie(muziek);
writeln(logfile);
writeln(logfile, 'EINDTOTAAL:', #9, aspiratiescore:0:6);
schrijfabc(aspiratie, bestandsnaammuz2);
close(infile);
close(logfile);

end.
```

E Voorbeeld van door Tabu-Music gegenereerde muziek



De score bedraagt 35.